(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENTAMT

(12) **Offenlegungsschrift**

(10) **DE 44 23 559 A1**

(51) Int. Cl.[6]:
G 06 F 12/08
G 06 F 15/16

(21) Aktenzeichen: P 44 23 559.3
(22) Anmeldetag: 5. 7. 94
(43) Offenlegungstag: 11. 5. 95

DE 44 23 559 A1

(30) Unionspriorität: (32) (33) (31)
09.11.93 US 149418

(71) Anmelder:
Hewlett-Packard Co., Palo Alto, Calif., US

(74) Vertreter:
Schoppe, F., Dipl.-Ing.Univ., Pat.-Anw., 82049 Pullach

(72) Erfinder:
Gupta, Rajiiv, Los Altos, Calif., US; Karp, Alan H., Sunnyvale, Calif., US

Prüfungsantrag gem. § 44 PatG ist gestellt

(54) Datenverbindungsverfahren und Vorrichtung für Multiprozessor-Computersysteme mit gemeinsamem Speicher

(57) Die vorliegende Erfindung offenbart ein Multiprozessor-Computersystem mit gemeinsam verwendetem Speicher, bei dem mehrere Prozessoren Kopien eines gemeinsam verwendeten Datenblocks in ihren lokalen Cache-Speichern zwischenspeichern können und unabhängig ihre zwischengespeicherten Kopien verändern können. Die zwischengespeicherten Kopien werden später in einem globalen Speicher mit dem gemeinsam verwendeten Datenblock verbunden. Mit jeder zwischengespeicherten Kopie wird ebenfalls eine Bitmaske, die aus einer Mehrzahl von Flags besteht, die Elementen der zwischengespeicherten Kopie zugeordnet sind, in den lokalen Speichern gespeichert. Eine lokale Speichersteuerung verfolgt durch Setzen der Bitmasken-Flags, die solchen Elementen zugeordnet sind, welche Elemente der zwischengespeicherten Kopien verändert wurden. Beim Verbinden werden lediglich veränderte Elemente der zwischengespeicherten Kopien in dem ursprünglichen Datenblock gespeichert, wie es durch die Bitmasken-Flags angezeigt ist.

DE 44 23 559 A1

Beschreibung

Diese Erfindung bezieht sich im allgemeinen auf Computersysteme, bei denen in einem Speicher gespeicherte Daten von mehreren Prozessoren global gemeinsam verwendet werden, und insbesondere bezieht sie sich auf Verfahren und eine Vorrichtung zum Beibehalten der Kohärenz von gemeinsamen Daten, die bei einzelnen Prozessoren gespeichert sind.

Bei Mehrprozessor-Computersystemen sind mehrere unabhängige Prozessoren miteinander verbunden, um die gleichzeitige Ausführung von mehreren Prozessen, die Teile eines einzelnen Programms darstellen, zu ermöglichen. Solche Computersysteme können verwendet werden, um mehrere getrennte Programme, die sich Daten oder Speichermittel teilen, auszuführen.

Multiprozessor-Systeme mit gemeinsamem Speicher, bei denen die Daten in einem Speicher gespeichert sind, auf den durch alle Prozessoren global zugegriffen werden kann, bieten den Vorteil eines flachen Speichermodells, das durch viele Programmierer bevorzugt wird. Bei Systemen mit einem flachen Speichermodell kann auf die Daten durch einzelne Prozessoren zugegriffen werden, die eine globale virtuelle Adressierung verwenden. Im Gegensatz dazu sind bei einigen Multiprozessor-Systemen mit verteiltem Speicher die Daten in Speichern gespeichert, auf die lediglich durch einzelne Prozessoren zugegriffen werden kann. Bei solchen Systemen kann ein Prozessor lediglich auf Daten in einem Speicher eines anderen Prozessor zugreifen, indem er direkt mit einem Programm, das auf dem anderen Prozessor abläuft, in Verbindung tritt. Die Aufgabe des Programmierers ist folglich komplexer, weil die Voraussetzung geschaffen werden muß, um Meldungen zwischen Prozessoren auszutauschen und den Ort der Daten in den Prozessoren zu verfolgen.

Bei vielen Multiprozessor-Systemen mit gemeinsamem Speicher ist jeder Prozessor mit einem lokalen Speicher oder Cache-Speicher zum schnelleren Speicherzugriff versehen. Weil der globale Speicher und die Prozessoren physikalisch voneinander entfernt ist, ist der direkte Zugriff auf Daten in dem globalen Speicher verhältnismäßig langsam. Der Cache-Speicher schafft eine schnellere Zwischenspeicherstufe zur vorübergehenden Speicherung von gemeinsamen Daten, die ein Prozessor benötigt. Wenn der Prozessor auf Daten eines Datenblocks in dem globalen Speicher zugreift, wird der gesamte Datenblock übertragen und in dem Cache-Speicher des Prozessors gespeichert. Der anfängliche Datenzugriff ist immer noch langsam, nachdem der Block aus dem globalen Speicher erhalten werden muß, aber nachfolgende Zugriffe auf den gleichen Datenblock können mit der schnelleren Rate des Cache-Speichers durchgeführt werden. Die meisten Datenzugriffe durch einen Prozessor sind temporär und räumlich lokalisiert. D.h. während einer gegebenen Zeitdauer wird ein Hauptanteil von Datenzugriffen durch einen Prozessor im selben Datenblock erfolgen. Durch das Zwischenspeichern im Cache-Speicher wird dementsprechend auf die Daten mit einer viel schnelleren Durchschnittsrate zugegriffen, als wenn alle Datenzugriffe direkt über den globalen Speicher durchgeführt würden.

Bei Multiprozessor-Systemen mit gemeinsamem Speicher, die Cache-Speicher verwenden, muß jedoch ein weiteres Problem beachtet werden, nämlich die Beibehaltung der Cache-Speicher-Kohärenz. Wenn mehrere Prozessoren Kopien desselben Datenblocks haben und die im Cache-Speicher gespeicherten Kopien durch einen oder mehrere der Prozessoren verändert werden, werden die in dem Cache-Speicher gespeicherten Datenblöcke inkonsistent sein. Bei der Ausführung der jeweiligen Prozesse durch die Prozessoren kann dies fehlerhafte Ergebnisse verursachen.

Eine mangelnde Cache-Speicher-Kohärenz tritt z. B. auf, wenn zwei Prozessoren in ihren jeweiligen Cache-Speichern Kopien desselben Datenblockes haben und beide dasselbe Datenelement in den Blöcken verändern. In einer solchen Situation sind die im Cache-Speicher gespeicherten Kopien des Prozessors zueinander und zu dem Datenblock im globalen Speicher inkonsistent. Abhängig davon, welche Kopie eines Prozessors den globalen Speicher zuletzt aktualisiert, wird das Datenelement in dem Datenblock im globalen Speicher den Wert haben, der durch einen Prozessor oder einen anderen geschrieben wurde (und nicht notwendigerweise den Wert, der tatsächlich zuletzt durch die Prozessoren geschrieben wurde). Dies wird manchmal als "Schreibe/Schreibe-Datenwettlauf" (write/write data race) bezeichnet.

Eine mangelnde Cache-Speicher-Kohärenz erfolgt ebenfalls, wenn ein erster Prozessor schreibt und ein zweiter Prozessor dasselbe Datenelement eines Datenblocks liest. Abhängig davon, ob die im Cache-Speicher des zweiten Prozessors gespeicherte Kopie des Datenblocks erhalten wird, bevor oder nachdem der Datenblock in dem globalen Speicher mit dem durch den ersten Prozessor geschriebenen Wert aktualisiert ist, wird der zweite Prozessor den ursprünglichen Wert des Datenelements oder den Wert, der durch den ersten Prozessor geschrieben wurde, lesen. Dies wird manchmal als "Lese/Schreibe-Datenwettlauf" bezeichnet.

Das Problem der Cache-Speicher-Kohärenz wurde ausgiebig untersucht und es wurde eine Anzahl von Lösungen vorgeschlagen. Eine Anzahl dieser Lösungen arbeitet durch Erzwingen einer Folgekonsistenz. Eine solche Lösung besteht darin, das Speichern von gemeinsamen Daten im Cache-Speicher einfach zu umgehen. Jeder Zugriff auf ein Datenelement, das mit anderen Prozessoren gemeinsam verwendet wird, wird direkt im globalen Speicher durchgeführt. Auf diese Art greifen die Prozessoren in einer strikten sequentiellen Art auf die gemeinsamen Daten zu. Diese Lösung weist erhebliche Geschwindigkeitsnachteile bei Systemen auf, bei denen direkte Zugriffe auf den globalen Speicher viel langsamer sind als Zugriffe auf in Cache-Speichern gespeicherten Daten, oder bei denen eine große Menge von gemeinsamen Daten vorhanden ist.

Eine weitere Variation der Folgekonsistenz, die hier als "ungültig machen beim Schreiben" bezeichnet wird, besteht darin, das gleichzeitige Speichern eines Datenblocks im Cache-Speicher durch mehrere Prozessoren zu ermöglichen, erfordert es aber, daß jeder Prozessor, der den Datenblock schreibt, ein Ungültig-Signal an alle anderen Prozessoren in dem System sendet. Die anderen Prozessoren müssen beim Empfangen des Ungültig-Signals eine aktualisierte Version des Datenblocks aus dem globalen Speicher anfordern, bevor sie weiter auf den Datenblock zugreifen. Eine weitere Variation der Folgekonsistenz, die hier als "Aktualisierung beim Schreiben" bezeichnet wird, erfordert es, daß der schreibende Prozessor jedesmal, wenn ein Schreiben durchgeführt wird, seine Veränderungen des Datenblocks an die anderen Prozessoren in dem System sendet.

Sowohl das "ungültig machen beim Schreiben"- und das "Aktualisieren beim Schreiben"-Schema sind jedoch weniger effizient. Bei beiden Schemata werden Ungül-

tig- oder Aktualisierungs-Meldungen unnötigerweise gesendet, wenn Datenblöcke fälschlicherweise gemeinsam verwendet werden. In der Situation, die hier als "falsche gemeinsame Verwendung" bezeichnet wird, speichert mehr als ein Prozessor denselben Datenblock im Cache-Speicher, aber auf ein Datenelement, das durch einen schreibenden Prozessor verändert wird, wird durch einen anderen Prozessor nicht zugegriffen (lesen oder schreiben) (d. h. der Datenblock wird gemeinsam verwendet, aber das Datenelement nicht). Solange kein Prozessor auf ein Datenelement, das durch einen anderen Prozessor verändert wurde zugreift, wird kein Konflikt erzeugt. Nachdem das falsche gemeinsame Verwenden für einen großen Anteil von Ungültig- und Aktualisierungs-Meldungen zutreffen kann, führen diese Schemata zu einem Anstieg von unnötigem Kommunikationsverkehr.

Der Verhaltenseinfluß einer falschen gemeinsamen Verwendung kann mit Cache-Speicher-Kohärenzlösungen gemildert werden, die es ermöglichen, daß die Prozessoren vorübergehend inkonsistente im Cache-Speicher gespeicherte Kopien eines Datenblocks beibehalten. Diese inkonsistenten im Cache-Speicher gespeicherten Kopien werden später an programmierten Synchronisationspunkten in Einklang gebracht (d. h. Aktualisierungen werden durchgeführt) (Synchronisationspunkte können z. B. durch die Verwendung von geeigneten Synchronisationsoperationen, wie z. B. Sperr- und Verriegelungs-Operationen, die in dem Programm angeordnet sind, eingestellt werden). Diese Synchronisationen müssen richtig programmiert sein, um die oben beschriebenen "Schreibe/Schreibe-" und "Lese/Schreibe-Datenwettläufe" zu vermeiden. Zwischen den Synchronisationen können mehrere Prozessoren jedoch verschiedene Abschnitte desselben Datenblocks in ihren jeweiligen Cache-Speichern verändern. Dieser Lösungsansatz, der hier als "verzögerte Konsistenz" bezeichnet ist, reduziert den Kommunikationsverkehr, der der falschen gemeinsamen Verwendung zugeordnet ist.

Eine verzögerte Konsistenzlösung wird durch A. Karp und V. Sarkar in "Data Merging for Shared Memory Multiprocessors", HP Laboratories Technical Report, November 1992 vorgeschlagen. Bei Karp-Sarkar-Datenverbinden wird es mehr als einem Prozessor ermöglicht, eine Kopie eines Datenblocks im globalen Speicher in seinem Cache-Speicher zu speichern und getrennte Abschnitte ihrer jeweiligen im Cache-Speicher gespeicherten Kopie unabhängig zu verändern. Jede im Cache-Speicher gespeicherte Kopie wird später an den globalen Speicher zurückübertragen und mit dem Datenblock verbunden. Das Verbinden der im Cache-Speicher gespeicherten Kopien und des Datenblocks im globalen Speicher wird unter Verwendung einer Bitmaske durchgeführt, die eine Bit-Flag für jedes Byte des Datenblocks im globalen Speicher enthält. Jede Flag der Bitmaske zeigt an, ob ein zugeordnetes Byte des Datenblocks im globalen Speicher bei einer Verbindung verändert wurde. Die Bitmaske, deren Flags anfänglich nicht gesetzt sind, wird dem Datenblock im globalen Speicher zugeordnet, wenn der Block zum ersten Mal durch irgendeinen Prozessor im Cache-Speicher gespeichert wird. Eine im Cache-Speicher gespeicherte Kopie, die durch einen Prozessor modifiziert wurde, die dann aus dem Cache-Speicher des Prozessors gelöscht wurde (um z. B. ihre Ersetzung durch eine Kopie eines weiteren Datenblocks zu ermöglichen), wird in den globalen Speicher zurückübertragen, um mit dem ursprünglichen Datenblock im globalen Speicher verbunden zu werden.

Jede modifizierte im Cache-Speicher gespeicherte Kopie wird mit dem ursprünglichen Datenblock durch Vergleichen jeder Flag der Bitmaske, die noch nicht gesetzt ist, des zugeordneten Bytes des ursprünglichen Datenblocks und des Bytes in einem entsprechenden Abschnitt der im Cache-Speicher gespeicherten Kopie verbunden. Wenn sich die Bytes unterscheiden, wurde das Byte in der im Cache-Speicher gespeicherten Kopie verändert. Das veränderte im Cache-Speicher gespeicherte Byte wird deshalb anstelle des Bytes des ursprünglichen Datenblocks gespeichert. Die Flag, die diesem Byte zugeordnet ist, wird ebenfalls gesetzt, um anzuzeigen, daß es verändert wurde. Wenn jedoch die Bytes identisch sind, bleibt die Flag ungesetzt und das unveränderte im Cache-Speicher gespeicherte Byte wird nicht im Datenblock im globalen Speicher gespeichert. Dementsprechend können zwei oder mehrere im Cache-Speicher gespeicherte Kopien des Datenblocks mit getrennten veränderten Abschnitten im globalen Speicher verbunden werden.

Das Karp-Sarkar-Datenverbindungssystem enthält ebenfalls einen Zähler für jeden Datenblock im globalen Speicher, der die Anzahl von Prozessoren nachverfolgt, die Kopien des Blocks in ihren Cache-Speichern haben. Der Zähler wird für jeden Prozessor, der eine Kopie des Blocks im Cache-Speicher speichert, erhöht. Die Prozessoren benachrichtigen den globalen Speicher immer dann, wenn eine Kopie des Datenblocks aus ihren Cache-Speichern gelöscht wird, sogar wenn dieser nicht verändert wurde, so daß der Zähler erniedrigt werden kann. Durch Nachverfolgen dieser Zahl kann die Bitmaske des Datenblocks reinitialisiert werden, wenn die Anzahl der Prozessoren, die Kopien des Datenblocks haben, auf Null zurückgeht.

Ein Nachteil des Karp-Sarkar-Datenverbindungsansatzes besteht darin, daß, sobald eine veränderte im Cache-Speicher gespeicherte Kopie mit einem Datenblock im globalen Speicher verbunden wurde, es keinem Prozessor gestattet werden kann, eine weitere Kopie des Blocks im Cache-Speicher zu speichern, bis die Verbindung vollständig ist (d. h. alle in Cache-Speichern gespeicherten Kopien des Blocks verbunden sind). Um solche weiteren Speicherungen des Blocks in Cache-Speichern nach dem Beginn des Verbindens zu verhindern, ist ein Aussetzungsmechanismus vorgesehen. Ein Aussetzungsbit ist dem Datenblock im globalen Speicher zugeordnet. Wenn eine im Cache-Speicher gespeicherte Kopie anfangs mit dem Block verbunden wird und eine weitere Kopie des Blocks im Cache-Speicher bleibt, wird das Aussetzungsbit gesetzt. Während das Aussetzungsbit gesetzt ist, wird jede Anfrage bezüglich eines Zugriffs auf den Block in einer Aussetzungswarteschlange angeordnet. Die Anfragen bleiben ausgesetzt, bis die Verbindung vollständig ist (d. h. bis der Zählwert der Prozessoren, die im Cache-Speicher gespeicherte Kopien haben, auf Null reduziert ist), woraufhin die Anfragen durchgeführt werden und das Bit gelöscht wird.

Das Verhindern eines weiteren Zugriffs auf einen Datenblock im globalen Speicher nach dem Beginn des Verbindens ist bei dem Karp-Sarkar-Datenverbindungsansatz notwendig, weil ein Prozessor ein bestimmtes Byte eines Datenblocks in dem globalen Speicher mehrere Male zwischen den Synchronisationspunkten in Übereinstimmung mit seinem Programm ordnungsgemäß verändern kann, ohne einen Datenwettlauf zu erzeugen. Zwischen solchen Veränderungen

muß die im Cache-Speicher gespeicherte Kopie des Prozessors des Blocks aus dem Cache-Speicher gelöscht werden, so daß der Cache-Speicher verwendet werden kann, um Kopien von anderen Blöcken zu speichern. Sobald jedoch die veränderte im Cache-Speicher gespeicherte Kopie gelöscht ist, wird diese mit dem Block im globalen Speicher verbunden und die Flag, die dem modifizierten Byte zugeordnet ist, wird gesetzt. Wenn es dem Prozessor ermöglicht wird, erneut eine Kopie des Blocks in seinem Cache-Speicher zu speichern und das Byte zu modifizieren, kann es nicht ordnungsgemäß mit dem Datenblock im globalen Speicher verbunden werden. (Die weitere Veränderung des Bytes wird in dem globalen Speicher nicht gespeichert, weil das Datenbit bereits gesetzt ist). Der Aussetzungsmechanismus ermöglicht es dem Prozessor, dasselbe Byte vor der nächsten Synchronisation im Cache-Speicher zu speichern und zu verändern, aber nur nachdem das Verbinden abgeschlossen ist und die dem Byte zugeordnete Flag gelöscht ist.

Es ist die Aufgabe der vorliegenden Erfindung, ein Datenverbindungsverfahren und eine Vorrichtung zu schaffen, die die Anforderung ausschließt, daß Zugriffe bezüglich des Zugriffs auf einen Datenblock ausgesetzt werden, sobald das Verbinden der Datenblöcke begonnen hat.

Diese Aufgabe wird durch ein Verfahren zur gemeinsamen Verwendung eines Speichers bei einem Computersystem mit mehreren Prozessoren nach Anspruch 1 und Anspruch 3, und durch ein Computersystem mit mehreren Prozessoren mit gemeinsamem Speicher nach Anspruch 10 gelöst.

Gemäß der vorliegenden Erfindung wird die Datenverbindung dadurch hergestellt, daß eine Bitmaske jeder im Cache-Speicher gespeicherten Kopie eines ursprünglichen Datenblocks zugeordnet wird. Die Bitmaske umfaßt eine Mehrzahl von Flags, eine für jedes Datenelement in der im Cache-Speicher gespeicherten Kopie, um anzuzeigen, welche im Cache-Speicher gespeicherte Kopie der Datenelemente verändert wurde. Die Bitmasken-Flags werden initialisiert, wenn die Kopie das erstemal im Cache-Speicher gespeichert wird. Wenn die Datenelemente der im Cache-Speicher gespeicherten Kopie verändert werden, werden deren zugeordnete Flags auf einen geschriebenen Wert eingestellt.

Wenn die im Cache-Speicher gespeicherte Kopie danach aus dem Cache-Speicher gelöscht wird, bestimmt die Bitmaske, welche Datenelemente der im Cache-Speicher gespeicherten Kopie im Datenblock im globalen Speicher gespeichert werden. Jede im Cache-Speicher gespeicherte Kopie des Datenelements, deren zugeordnete Flag anzeigt, daß diese verändert wurde, wird in dem Datenblock im globalen Speicher gespeichert. Während des Verbindens werden dementsprechend Datenelemente der im Cache-Speicher gespeicherten Kopie an den entsprechenden Stellen des ursprünglichen Datenblocks gespeichert, wenn die im Cache-Speicher gespeicherte Kopie der Datenelemente verändert ist, und nicht gespeichert, wenn die ursprünglichen Datenblockelemente unverändert bleiben. Durch ein solches Schema kann ein Prozessor eine Kopie eines Blocks im globalen Speicher erfolgreich im Cache-Speicher speichern, diese verbinden und wiederum eine Kopie des Blocks im Cache-Speicher speichern, ohne darauf zu warten, daß der Verbinder fertig wird.

Bei einem Ausführungsbeispiel der Erfindung sind die im Cache-Speicher gespeicherten kopierten Datenelemente, denen die Bitmasken-Flags zugeordnet sind, Bytes. (Die Bezeichnung "Byte" wird hier verwendet, um ein Mehr-Bit-Datenelement zu bezeichnen, das im allgemeinen, aber nicht notwendigerweise, sieben oder acht binäre Stellen ("Bits") hat). Bei einem weiteren Ausführungsbeispiel der Erfindung sind die im Cache-Speicher gespeicherten kopierten Datenelemente, denen die Bitmasken-Flags zugeordnet sind, Datenwörter. (Die Bezeichnung "Datenwort" wird hier verwendet, um eine Einheit von Daten zu bezeichnen, die aus einer ganzzahligen Anzahl von Bytes besteht, im allgemeinen, aber nicht notwendigerweise, vier Bytes). Durch Beibehalten einer Bitmaske mit Flags, die lediglich den Wörtern der im Cache-Speicher gespeicherten Kopien zugeordnet sind, ist weniger Speicherraum in den Cache-Speichern der Prozessoren zum Speichern der Bitmasken erforderlich.

In Übereinstimmung mit einem weiteren Aspekt der Erfindung ist bei einem Ausführungsbeispiel der Erfindung, bei dem die Bitmasken-Flags den im Cache-Speicher gespeicherten kopierten Datenwörtern zugeordnet sind, ebenfalls eine Teilwort-Speicher-Flag der im Cache-Speicher gespeicherten Kopie zugeordnet. Die Teilwort-Speicher-Flag wird initialisiert, wenn die Kopie im Cache-Speicher gespeichert wird und nachfolgend gesetzt, wenn der Prozessor eine Teilwort-Speicher-Operation bezüglich der im Cache-Speicher gespeicherten Kopie durchführt (d. h. eine Operation, die weniger als das gesamte Wort speichert). Die im Cache-Speicher gespeicherte Kopie wird dann unter Verwendung einer Bitmaske, die eine Flag für jedes Byte des Datenblocks im globalen Speicher hat, wie beim Karp-Sarkar-Datenverbindungsansatz verbunden. Die Bitmaske des globalen Speichers kann in einem dynamisch zugeordneten Speicher gespeichert werden. Dieser Aspekt der Erfindung reduziert den Cache-Speicher-Speicherplatzmehraufwand, während Teilwort-Speicherzugriffe ermöglicht werden.

Gemäß eines weiteren Aspekts der Erfindung sind eine Lese-Bitmaske und eine Schreibe-Bitmaske in dem Cache-Speicher enthalten. Jede der Lese- und Schreibe-Bitmasken enthält eine Flag für jedes im Cache-Speicher gespeicherte kopierte Datenelement, die gesetzt wird, wenn das zugeordnete Datenelement gelesen bzw. geschrieben wird. Nachdem die im Cache-Speicher gespeicherte Kopie mit dem Datenblock im globalen Speicher verbunden ist, werden die Bitmasken gespeichert, so daß das Auftreten eines Lese/Schreibe- oder Schreibe/Schreibe-Datenwettlaufes zur Fehlerbeseitigung oder aus anderen Gründen identifiziert werden kann.

Gemäß einem weiteren Aspekt der Erfindung ist jedem Datenblock im globalen Speicher eine Sperr-Flag zugeordnet. Die Sperr-Flag wird bei der Anfrage eines sperrenden Prozessors gesetzt. Wenn die Sperr-Flag gesetzt ist, werden weitere Anfragen bezüglich des Zugriffs auf den Datenblock in einer Anfrageaussetzungswarteschlange angeordnet. Die Sperr-Flag kann lediglich auf die Anfrage des sperrenden Prozessors gelöscht werden, woraufhin die ausgesetzten Anfragen verarbeitet werden.

Wiederum gemäß eines weiteren Aspekts der Erfindung werden Reduktionsoperationen bezüglich der Daten im globalen Speicher durchgeführt. Solche Reduktionsoperationen bezüglich der Daten im globalen Speicher können durch eine Steuerung des globalen Speichers durchgeführt werden.

Gemäß einem weiteren Aspekt der Erfindung ist in einem Computersystem mit mehreren globalen Speichereinheiten eine Kopie eines Datenblocks, der in ei-

nem ersten globalen Speicher gespeichert ist, ebenfalls in einem zweiten globalen Speicher gespeichert. Die Prozessoren können dann im Cache-Speicher gespeicherte Kopien des Datenblocks aus einer der globalen Speichereinheiten erhalten. Wenn alle im Cache-Speicher gespeicherten Kopien wieder mit den Kopien des Datenblocks im globalen Speicher verbunden wurden, werden die Kopien im globalen Speicher ebenfalls verbunden. Bevorzugterweise sind die mehreren globalen Speichereinheiten in einer Hierarchie angeordnet und die Kopien in dem globalen Speicher werden mit einer nächst höheren Stufe des globalen Speichers verbunden. Dieser Aspekt kann angewendet werden, wenn auf den Datenblock eines globalen Speichers durch eine große Anzahl von Prozessoren zugegriffen wird, um die Überlastung dieses globalen Speichers zu verhindern, während die anderen nicht verwendet werden.

Bevorzugte Ausführungsbeispiele der vorliegenden Erfindung werden nachfolgend unter Bezugnahme auf die beiliegenden Zeichnungen genauer beschrieben. Es zeigen:

Fig. 1 eine schematische Darstellung eines Multiprozessor-Computersystems mit gemeinsamem Speicher gemäß der vorliegenden Erfindung;

Fig. 2 eine Darstellung gemäß einem ersten Ausführungsbeispiel der Erfindung, die die Beziehung zwischen den Flags in einer Schreibe-Bitmaske und Datenelementen eines im Cache-Speicher gespeicherten Datenblocks, der in einem lokalen Speicher des Computersystems aus Fig. 1 gespeichert ist, darstellt;

Fig. 3 eine Darstellung gemäß einem zweiten, dritten und neunten Ausführungsbeispiel der Erfindung, die die Beziehung der Schreibe-Bitmasken-Flags zu Datenelementen eines im Cache-Speicher gespeicherten Datenblocks, der in dem lokalen Speicher des Computersystems aus Fig. 1 gespeichert ist, darstellt;

Fig. 4 eine Darstellung, gemäß einem dritten und vierten Ausführungsbeispiel der Erfindung, die eine globale Schreibe-Bitmaske, eine Aussetzungs/Sperr-Flag, eine Aussetzungswarteschlange und einen Zählstand, die einem Datenblock zugeordnet sind und in einem globalen Speicher des Computersystems aus Fig. 1 gespeichert sind, darstellt;

Fig. 5 eine Darstellung gemäß einem fünften Ausführungsbeispiel der Erfindung, die lokale und globale Schreibe- und Lese/Bitmasken darstellt, die in dem lokalen Speicher und dem globalen Speicher des Computersystems aus Fig. 1 zur Identifikation von Datenfehlern gespeichert sind;

Fig. 6 eine Darstellung gemäß einem siebten Ausführungsbeispiel der Erfindung, die eine hierarchische Anordnung von Prozessoren und globalen Speichereinheiten in dem Computersystem aus Fig. 1 darstellt;

Fig. 7 eine Darstellung gemäß einem achten Ausführungsbeispiel der Erfindung, die eine globale Schreibe-Bitmaske, eine spezielle Verbinder-Flag und eine Tabelle darstellt, die einem Datenblock zugeordnet sind und in einem globalen Speicher des Computersystems aus Fig. 1 gespeichert sind; und

Fig. 8 ein verallgemeinertes Blockdiagramm gemäß einem neunten Ausführungsbeispiel der Erfindung, das die internen Register und Bit-Veränderungs-Flags einer CPU in dem Computersystem aus Fig. 1 darstellt.

In Fig. 1 ist ein erstes Ausführungsbeispiel der vorliegenden Erfindung dargestellt, bei dem ein Multiprozessor-Computersystem 20 mit gemeinsamem Speicher eine Mehrzahl von Prozessoren 22—24 und eine oder mehrere globale Speichereinheiten 26, 27 umfaßt. Ein Verbindungskommunikationsmedium 28, wie z. B. ein Bus, ein Kreuzschienenwähler, ein mehrstufiges Netzwerk, ein lokales Netzwerk oder ähnliches verbindet die Prozessoren 22—24 und die globalen Speichereinheiten 26, 27. Das Kommunikationsmedium 28 wird verwendet, um Daten zwischen den Prozessoren 22—24 und den globalen Speichereinheiten 26, 27 zu übertragen.

Jede globale Speichereinheit 26, 27 hat eine globale Speichersteuerung 30, 31 und einen globalen Speicher 32, 33. Die globalen Speicher 32, 33 stellen den Speicherplatz für Daten bereit, die durch die Prozessoren 22—24 gemeinsam verwendet werden sollen und können ebenfalls Daten speichern, die sich auf die Datenverbindung beziehen. Die globalen Speichersteuerungen 30, 31 reagieren auf globale Speicheranfragen der Prozessoren und führen Datenverbindungsoperationen aus. Die globalen Speichersteuerungen 30, 31 können als Zustandsmaschine ausgeführt sein. Die Steuerung 30, 31 sind jedoch bevorzugterweise mit allgemeinen Verarbeitungsschaltungen ausgeführt, die programmiert sind, um gemäß der Erfindung wirksam zu sein.

Jeder der Prozessoren 20—24 hat eine zentrale Verarbeitungseinheit 36—38, eine lokale Speichersteuerung 40—42 und einen lokalen oder Cache-Speicher 44—46. Die Prozessoren 22—24 führen einzeln Prozesse in ihren jeweiligen zentralen Verarbeitungseinheiten 36—38 aus, die einen Teil einer gemeinsamen programmierten Aufgabe darstellen. Wenn ein Prozeß, der in einer zentralen Verarbeitungseinheit 36—38 eines Prozessors ausgeführt wird, Zugriff auf gemeinsam verwendete Daten, die in einer der globalen Speichereinheiten 26, 27 gespeichert sind, benötigt, gibt die Lokalspeichersteuerung 40—42 des Prozessors eine Datenabfrage an die globale Speichereinheit aus. Daten, die durch einen der Prozessoren 22—24 von den globalen Speichereinheiten 26, 27 erhalten werden, werden in dessen lokalen Speichern 44—46 gespeichert.

Die gemeinsamen Daten, die in den globalen Speichern 32, 33 gespeichert sind, umfassen eine Mehrzahl von Datenblöcken, wobei jeder Datenblock im allgemeinen aus einer Anzahl von Bytes besteht. Um die durchschnittliche Zeitdauer zu reduzieren, die erforderlich ist, um auf die gemeinsamen Daten zuzugreifen, wird es den Prozessoren 22—24 ermöglicht, Kopien der gemeinsamen Datenblöcke in ihren lokalen Speichern 44—46 zwischenzuspeichern. Bevorzugterweise sind die lokalen Speicher 44—46 Halbleiterspeicher vom Typ des wahlfreien Zugriffs, die schnellere Zugriffszeiten, die der Befehlsausführungsgeschwindigkeit der zentralen Verarbeitungseinheiten 36—38 entsprechen, bereitstellen können. Die globalen Speichermodule 32, 33 können ebenfalls relativ schnelle Speicher vom Halbleitertyp sein, können aber langsamere Massenspeichergeräte, wie z. B. Datenspeichergeräte, die optische oder magnetische Medien verwenden, einschließen. Wenn dementsprechend eine zentrale Verarbeitungseinheit 36—38 eines Prozessors 22—24 Zugriff auf gemeinsame Daten in einer globalen Speichereinheit 26—27 erfordert, gibt die lokale Speichersteuerung 40—42 des Prozessors eine Datenabfrage an die globale Speichereinheit aus. Die globale Speichersteuerung 30, 31 in der globalen Speichereinheit reagiert durch Übertragen einer Kopie des Datenblocks, der die angeforderten Daten enthält, an den Prozessor. Die lokale Speichersteuerung 40, 42 des Prozessors speichert dann die Datenblockkopie in dem lokalen Speicher 44—46 des Prozessors.

Bei dem Computersystem 20 können mehr als ein

Prozessor 20—24 Kopien desselben Datenblocks im globalen Speicher in ihren lokalen Speichern 44—46 zwischenspeichern und Abschnitte solcher Kopien unabhängig verändern. Als ein Ergebnis der unabhängigen Veränderung der zwischengespeicherten Kopien können sich die Kopien voneinander unterscheiden. Diese Unterschiede zwischen den zwischengespeicherten Kopien werden durch die im folgenden beschriebene Datenverbindung gelöst.

Nachdem die Kopien eines gemeinsamen Datenblocks durch mehrere Prozessoren in dem Computersystem 20 zwischengespeichert und verändert werden können, können Lese/Schreibe- und Schreibe/Schreibe-Datenwettläufe erzeugt werden. Diese Wettläufe werden bei dem Computersystem 20 durch Programmierung der Prozesse, die auf den Prozessoren 20—24 ablaufen, mit geeigneter Synchronisation vermieden. Um Schreibe/Schreibe-Datenwettläufe zu vermeiden, müssen die Prozesse richtig programmiert sein, so daß keine zwei Prozessoren ein gleiches Datenelement zwischen programmierten Synchronisationspunkten verändern können. Der Programmierer sollte ebenfalls sicherstellen, daß kein Prozessor ein Datenelement des gemeinsamen Blocks liest, der ebenfalls durch einen anderen Prozessor zwischen programmierten Synchronisationspunkten verändert (d. h. geschrieben) wurde. Daß mehr als ein Prozessor ein gleiches gemeinsames Datenelement zwischen Synchronisationen verändert, tritt in dem Computersystem 20 folglich lediglich als ein Ergebnis eines Fehlers des Programmierers auf. Die Synchronisation der Zugriffe auf ein gemeinsames Datenelement kann z. B. durch herkömmliche Sperr- oder Verriegelungs-Befehl oder anderen damit verbundenen Mechanismen durchgeführt werden.

Wie in **Fig. 2** dargestellt ist, wird die Datenverbindung gemäß dem ersten Ausführungsbeispiel der Erfindung durch Nachverfolgen, welche Datenelemente einer zwischengespeicherten Kopie verändert wurden, erreicht. Wenn die lokale Speichersteuerung 40 (**Fig. 1**) eine Kopie 54 eines Datenblocks (der "ursprüngliche Datenblock") aus der globalen Speichereinheit 26 in dem lokalen Speicher 44 speichert, speichert bei dem Computersystem 20 die Steuerung ebenfalls eine Schreibe-Bitmaske 56 in dem lokalen Speicher. Die Schreibe-Bitmaske 56 besteht aus einer Mehrzahl von Flags 60—64, eine für jedes einer Mehrzahl von Datenelementen 70—74 in der Kopie 54. Die Flags 60—64 sind in einer 1 : 1-Beziehung den Datenelementen 70—74 zugeordnet, wie es durch die Pfeile in **Fig. 2** dargestellt ist. Bei dem Ausführungsbeispiel, das in **Fig. 2** dargestellt ist, ist jede Flag ein einzelnes Bit, kann aber bei alternativen Ausführungsbeispielen mehrere Bits umfassen. Bei dem dargestellten Ausführungsbeispiel sind die Datenelemente, die den Bitmasken-Flags 1 : 1 zugeordnet sind, ebenfalls Bytes. Wie es jedoch in dem Ausführungsbeispiel, das in **Fig. 3** dargestellt ist und das im folgenden beschrieben wird, gezeigt ist, können die Datenelemente, die den Bitmasken-Flags 1 : 1 zugeordnet sind, Wörter oder Dateneinheiten mit irgendwelchen anderen Größen sein, die zusammen die zwischengespeicherte Kopie bilden.

Die lokale Speichersteuerung 40 verfolgt durch Setzen der Flags 60—64, die den Datenelementen, die verändert sind, zugeordnet sind, ob Datenelemente 70—74 der zwischengespeicherten Kopie 54 verändert wurden. Die lokale Speichersteuerung 40 setzt die Flags 60—64 anfänglich auf einen nicht-geschriebenen Wert (d. h. ein Wert, der anzeigt, daß seine zugeordneten Datenelemente 70—74 nicht verändert wurden). Bei dem dargestellten Ausführungsbeispiel ist der nicht-geschriebene Wert der einzelnen Bit-Flags 60—64 Null. Wenn die zentrale Verarbeitungseinheit 36 später irgendeines der Datenelemente 70—74 verändert, setzt die lokale Speichersteuerung 40 die Flags, die solchen veränderten Datenelementen zugeordnet sind, auf einen veränderten Wert (z. B. Eins bei dem dargestellten Ausführungsbeispiel). Wenn die zentrale Verarbeitungseinheit 36 z. B. die Datenelemente 71 und 72 durch Schreiben dieser Elemente verändert, werden die jeweiligen zugeordneten Flags 61, 62 auf Eins gesetzt. Als ein Ergebnis zeigen die Bitmasken-Flags 60—64 an, welches der Datenelemente 70—74 der zwischengespeicherten Kopie verändert wurde.

Wenn die zwischengespeicherte Kopie 54 aus dem lokalen Speicher gelöscht wird, überträgt die lokale Speichersteuerung 40 die Bitmaske 56 und die zwischengespeicherte Kopie 54 an die globale Speichereinheit 26. Das Löschen der zwischengespeicherten Kopie 54 tritt z. B. auf, wenn die lokale Speichersteuerung 40 die zwischengespeicherte Kopie mit einer anderen Datenblockkopie ersetzt, so daß die zentrale Verarbeitungseinheit 36 auf Daten in der anderen Datenblockkopie zugreifen kann. Einige Synchronisationsmechanismen können ebenfalls das Löschen der zwischengespeicherten Kopie 54 durch Senden eines Ungültig-Signals an die lokale Speichersteuerung 40 auslösen. Bei einigen Ausführungsbeispielen kann die lokale Speichersteuerung 40 lediglich so viele Informationen übertragen, die zur Datenverbindung notwendig sind, wodurch der Kommunikationsmehraufwand reduziert wird, die Steuerung überträgt z. B. die Bitmaske 56 und lediglich diejenigen Datenelemente der zwischengespeicherten Kopie 54, die verändert wurden.

In der globalen Speichereinheit 26 verwendet die globale Speichersteuerung 30 (**Fig. 1**) die Bitmaske 56 beim Verbinden der zwischengespeicherten Kopie 54 mit ihrem ursprünglichen Datenblock. Für jede der Flags 60—64 der Bitmaske 54, die auf den veränderten Wert eingestellt ist, speichert die globale Speichersteuerung 30 die zugeordneten Datenelemente 70—74 der zwischengespeicherten Kopie in dem ursprünglichen Datenblock. Nachdem z. B. die Flags 61, 62 in der Darstellung von **Fig. 2** auf Eins gesetzt sind (d. h. den veränderten Wert), werden die zugeordneten Datenelemente 71, 72 an entsprechende Stellen des ursprünglichen Datenblocks gespeichert. Dies bewirkt eine konsistente Verbindung in dem Datenblock des globalen Speichers mit mehreren zwischengespeicherten Kopien, nachdem lediglich die veränderten Abschnitte der zwischengespeicherten Kopien den Datenblock aktualisieren und Programmierungshindernisse mehr als einem Prozessor verbieten, denselben Abschnitt des Datenblocks zu verändern.

Wie in der Beschreibungseinleitung beschrieben wurde, besteht ein Nachteil des bekannten Karp-Sarkar-Datenverbindungsansatzes darin, daß ein Prozessor daran gehindert wird, ein Element eines zwischengespeicherten Datenblocks zu verändern, den Block zu verbinden und den Datenblock wieder zwischenzuspeichern, bevor alle Kopien des Blocks verbunden wurden. Dies liegt daran, daß der Prozessor wiederum dasselbe Element im Datenblock verändern könnte und eine solche weitere Veränderung würde durch eine Datenverbindung nach Karp-Sarkar nicht im globalen Speicher gespeichert werden. Beim Karp-Sarkar-Datenverbinden verfolgt eine Bitmaske im globalen Speicher, wel-

che Elemente des globalen Blocks verändert sind. Elemente einer zwischengespeicherten Kopie werden lediglich gespeichert, wenn sie sich von einem bisher unveränderten Element an einer entsprechenden Stelle des globalen Blocks unterscheiden. Genauer gesagt aktualisiert eine Veränderung eines Datenelements durch einen Prozessor in dessen erster zwischengespeicherter Kopie eines Datenblocks das Element an der entsprechenden Stelle des globalen Blocks und setzt eine zugeordnete Flag. Sobald die Flag gesetzt ist, können keine weiteren Änderungen des Elements den globalen Block aktualisieren, bis alle zwischengespeicherten Kopien des Blocks verbunden sind und die Flag zurückgesetzt ist. Aus diesem Grund verwendet das Karp-Sarkar-Datenverbindungssystem einen Aussetzungsmechanismus, der jeden Prozessor daran hindert, den Datenblock nach seiner ersten Verbindung zwischenzuspeichern, bis alle bisher zwischengespeicherten Kopien des Blocks ebenfalls verbunden wurden.

Mehrere Modifikationen eines Datenelements durch denselben Prozessor zwischen Synchronisationen sind jedoch bei Systemen, die eine verzögerte Konsistenz verwenden, zulässig. Lediglich die Modifikation desselben Datenelements durch mehr als einen Prozessor wird als Fehler betrachtet. Der Karp-Sarkar-Datenverbindungsansatz bewirkt dann eine unerwünschte, willkürlich lange Verzögerung der Prozessoren, die dasselbe Element eines Datenblocks wiederum verändern müssen, nachdem ihre erste zwischengespeicherte Kopie des Blocks gelöscht wurde.

Bei dem Datenverbindungsansatz der vorliegenden Erfindung kann ein Prozessor einen Datenblock beliebig oft zwischenspeichern und verändern. Alle veränderten Elemente einer zwischengespeicherten Kopie werden immer gespeichert. Deshalb ist es nicht notwendig, den Prozessor daran zu hindern, Kopien eines Datenblocks nach dem Beginn der Verbindung, aber bevor diese abgeschlossen ist, zwischenzuspeichern. Folglich wird kein Aussetzungsmechanismus benötigt.

Gemäß einem zweiten Ausführungsbeispiel, wie in Fig. 3 dargestellt ist, verfolgen die lokalen Speichersteuerungen 40—42 in dem Computersystem 20 unter Verwendung einer "Wort"-Schreibe-Bitmaske 88, die Flags 90—92 aufweist, die in einer 1 : 1-Beziehung den Wörtern 80—82 zugeordnet sind, welche Wörter 80—82 einer zwischengespeicherten Kopie 86 modifiziert sind. Jedes Wort 80—82 besteht aus einer ganzzahligen Anzahl von Bytes 96—107 (z. B. vier Bytes pro Wort bei dem Ausführungsbeispiel, das in Fig. 3 dargestellt ist). Um nachzuverfolgen, welche Wörter verändert sind, sind die Flags 90—92 anfänglich auf einen nicht-geschriebenen Wert eingestellt. Wenn die Wörter 80—82 in der zwischengespeicherten Kopie 86 verändert sind, werden dann ihre jeweiligen zugeordneten Flags 90—92 auf einen geschriebenen Wert gesetzt. Wie es z. B. in Fig. 3 dargestellt ist, wird die Flag 91, wenn die Flags 90—92 anfänglich auf Null gesetzt sind, auf Eins gesetzt, wenn das Wort 81 verändert ist, wodurch angezeigt ist, daß es verändert wurde.

Durch Zuordnen der Flags 90—92 zu Wörtern der zwischengespeicherten Kopie wird der Speicherplatzmehraufwand, der der Nachverfolgung von Veränderungen von zwischengespeicherten Kopien zugeordnet ist, bei dem zweiten Ausführungsbeispiel reduziert. Das erste in Fig. 2 gezeigte Ausführungsbeispiel, bei dem eine Bitmasken-Flag für jedes Byte der zwischengespeicherten Kopie beibehalten wird, erfordert durch Vergleich etwa viermal mehr Speicherplatz für die Bitmasken-Flags. Die Beibehaltung einer Flag pro Wort ist besonders vorteilhaft, wenn die zentralen Verarbeitungseinheiten 36—38 in Inkrementen der Wortgröße auf die Daten zugreifen. Bei solchen zentralen Verarbeitungseinheiten, die weit verbreitet sind, ist die Nachverfolgung, welche Bytes 96—107 in der zwischengespeicherten Kopie 86 verändert sind, mittels einer Flag pro Byte in der Bitmaske 88 redundant.

Bei einem alternativen ("dritten") Ausführungsbeispiel der Erfindung, das in Fig. 3 dargestellt ist, das eine Wort-Schreibe-Bitmaske 88 beibehält, ist eine Teilwort-Speicher-Flag 110 der zwischengespeicherten Kopie 26 zugeordnet, um Teilwort-Speicher-Operationen anzupassen. Einige zentrale Verarbeitungseinheiten 36—38, die normalerweise auf die Daten durch das Wort zugreifen, sind ebenfalls fähig, Operationen durchzuführen (im folgenden "Teilwort-Speicher-Operationen" genannt), die Daten in Inkrementen von weniger als einem Wort verändern (z. B. die lediglich ein Byte zu einem Zeitpunkt verändern). Bei einem Computersystem 20, das solche zentrale Verarbeitungseinheiten verwendet, ist es wünschenswert, die Datenverbindung auf der Wortstufe normal durchzuführen, um den Speicherplatz- und den Übertragungs-Mehraufwand zu reduzieren, wobei jedoch eine Datenverbindung auf der Byte-Stufe vorgesehen ist, wenn Teilwort-Speicher-Operationen verwendet werden.

Wenn die zwischengespeicherte Kopie 86 dementsprechend in dem lokalen Speicher 44 gespeichert ist, speichert die lokale Speichersteuerung 40 ebenfalls die Teilwort-Speicher-Flag 110 in dem lokalen Speicher 44. Wenn die Flag 110 gespeichert ist, wird diese anfänglich auf einen "keine Teilwort-Speicherung aufgetreten"-Wert gesetzt, der anzeigt, daß keine Teilwort-Speicher-Operationen bezüglich der zwischengespeicherten Kopie 86 durchgeführt wurden. Zu solchen Zeitpunkten, zu denen die zentrale Verarbeitungseinheit eine Teilwort-Speicher-Operation durchführt, setzt die lokale Speichersteuerung 40 die Teilwort-Speicher-Flag auf einen "Teilwort-Speicherung aufgetreten"-Wert, der ein solches Auftreten anzeigt. Die Steuerung 40 setzt ebenfalls die Wort-Schreibe-Bitmasken-Flag, die dem Wort, das durch die Teilwort-Speicher-Operation verändert wurde, zugeordnet ist, auf den geschriebenen Wert. Bei dem dargestellten Ausführungsbeispiel ist die Teilwort-Speicher-Flag ein einzelnes Bit, der "keine Teilwort-Speicherung aufgetreten"-Wert ist Null und der "Teilwort-Speicherung aufgetreten"-Wert ist Eins. Wenn die zentrale Verarbeitungseinheit 36 z. B. eine Teilwort-Speicher-Operation durchführt, die Daten in das Byte 102 schreibt, das ein Teil des Worts 81 ist, setzt die lokale Speichersteuerung die Teilwort-Speicher-Flag 110 auf Eins und setzt ebenfalls die Schreibe-Bitmaske-Flag 91, die dem Wort 81 zugeordnet ist, auf Eins. Der Zustand der Wort-Bitmasken- und Teilwort-Speicher-Flags nach einer Teilwort-Speicher-Operation verändert das Byte 102, wie es in Fig. 3 dargestellt ist.

Wenn die zwischengespeicherte Kopie 86 nachfolgend aus dem lokalen Speicher 44 gelöscht wird, überträgt die lokale Speichersteuerung 40 die zwischengespeicherte Kopie 86, die zwischengespeicherte Kopie der Schreibe-Bitmaske 88 und die Teilwort-Speicher-Flag 110 an die globale Speichereinheit 26, bei der der ursprüngliche Datenblock 118 gespeichert ist. Wenn die Teilwort-Speicher-Flag 110 auf den "keine Teilwort-Speicherung aufgetreten"-Wert eingestellt ist, wird die gespeicherte Kopie 86 mit dem ursprünglichen Datenblock 118 durch Speichern jedes Wortes 80—82, dessen

zugeordnete Flag 90—92 in der Wort-Schreibe-Bitmaske 88 auf den geschriebenen Wert gesetzt ist, an den entsprechenden Stellen des ursprünglichen Datenblocks gespeichert. Wenn jedoch die Teilwort-Speicher-Flag 110 auf den "Teilwort-Speicherung aufgetreten"-Wert gesetzt ist, werden lediglich die veränderten Bytes der zwischengespeicherten Kopie 36 verbunden.

Um lediglich die veränderten Bytes zu verbinden, speichert die globale Speichersteuerung 30 eine Teilwort-Schreibe-Bitmaske in dem globalen Speicher, die im allgemeinen aus einer Flag für jedes Byte des ursprünglichen Datenblocks besteht, wobei jede Flag anfänglich auf einen nicht-geschrieben Wert gesetzt ist. Für jedes Wort 80—82 der zwischengespeicherten Kopie 86, deren zugeordnete Wort-Schreibe-Bitmasken-Flag 90—92 auf den geschriebenen Wert gesetzt ist, vergleicht die globale Speichersteuerung 30 die Bytes eines solchen Wortes mit den Bytes an den entsprechenden Stellen des ursprünglichen Datenblocks. Wenn sich die Bytes unterscheiden und die Teilwort-Schreibe-Bitmasken-Flag, die dem ursprünglichen Blockbyte zugeordnet ist, immer noch auf den nicht-geschriebenen Wert gesetzt ist, speichert die globale Speichersteuerung 30 die zwischengespeicherte Kopie des Bytes anstelle des ursprünglichen Datenblockbytes und setzt die Flag in der globalen Schreibe-Bitmaske, die dem ursprünglichen Datenblockbyte zugeordnet ist, auf den geschriebenen Wert. Zusätzlich werden Kopien des Datenblocks, die in anderen Prozessoren 23—24 zwischengespeichert sind, nachfolgend auf eine ähnliche Art und Weise verbunden.

Ein ursprünglicher Datenblock 118 der zwischengespeicherten Kopie 86 (Fig. 3) enthält z. B., wie es in Fig. 4 dargestellt ist, Wörter 120—122 an Stellen, die denjenigen der Wörter 80—82 (Fig. 3) entsprechen. Wenn die Teilwort-Speicher-Flag 110 (Fig. 3) auf Eins gesetzt ist, wodurch angezeigt ist, daß eine Teilwort-Speicher-Operation bezüglich der Kopie 86 durchgeführt wurde, werden die Kopie 86 und der Block 118 unter Verwendung einer globalen Schreibe-Bitmaske 124 verbunden. Die Bitmaske 124 umfaßt eine Flag 126—137 für jedes Byte 146—157 des Blocks 118. Die Verbindung wird durch Vergleichen der Bytes 100—103 (Fig. 3) jeder veränderten, zwischengespeicherten Kopie des Wortes 81 (angezeigt durch die zugeordnete Flag 91, die auf Eins gesetzt ist) mit denjenigen Bytes 150—153 an den entsprechenden Stellen des Blocks 118 bewirkt, deren zugeordnete Flags 130—133 immer noch auf Null gesetzt sind (wodurch angezeigt ist, daß sie noch nicht verändert sind). Die Bytes 100—103, die sich von den Bytes 150—153 unterscheiden, deren zugeordnete Flags 130—133 auf Null gesetzt sind, werden gespeichert und die zugeordneten Flags werden auf Eins gesetzt.

Bevorzugterweise ordnet die globale Speichersteuerung 30 der Teilwort-Schreibe-Bitmaske dynamisch Speicherplatz aus einem freien Speicherabschnitt des globalen Speichers zu. Dies dient dazu, den Speicherplatzmehraufwand zu minimieren, nachdem eine Teilwort-Schreibe-Bitmaske für einen Datenblock in dem globalen Speicher nur gespeichert wird, wenn sie zur Datenverbindung erforderlich ist.

Mit dieser Art der Teilwort-Datenverbindung wird ein verändertes Byte in einer zwischengespeicherten Kopie in dem globalen Speicher nur gespeichert, wenn das ursprüngliche Datenblockbyte mit einer entsprechenden Position nicht bereits verändert wurde. Aus diesem Grund muß derselbe Prozessor daran gehindert werden, eine Kopie des Blocks wiederum zwischenzuspeichern und dasselbe Byte wiederum zu modifizieren, bis die Verbindung vollständig ist. Bevorzugterweise werden die Prozessoren am erneuten Zwischenspeichern eines Datenblocks, nachdem eine Teilwort-Datenverbindung des Blocks begonnen hat, durch Aussetzen jeglicher weiterer Datenabfragen für den Block, bis die Verbindung vollständig ist, gehindert.

Um das Aussetzen von Datenabfragen vorzusehen, speichert die globale Speichersteuerung 30 eine Aussetzungs-Flag 160, siehe Fig. 4, für jeden Datenblock 118 in dem globalen Speicher. Anfänglich ist die Aussetzungs-Flag 160 auf einen Nicht-Aussetzungs-Wert gesetzt, der in einem Ausführungsbeispiel der Erfindung, das eine Ein-Bit-Aussetzungs-Flag verwendet, bevorzugterweise Null ist. Wenn eine Teilwort-Datenverbindung des Datenblocks 118 beginnt, wird ihre zugeordnete Aussetzungs-Flag 160 auf einen Aussetzungs-Wert gesetzt, bevorzugterweise Eins. Wenn die Aussetzungs-Flag 160 auf den Aussetzungs-Wert gesetzt ist, speichert die globale Speichersteuerung 30 die Anfrage in einer Aussetzungswarteschlange 162, wenn eine Datenabfrage empfangen wird. Die Aussetzungswarteschlange 162 ist eine Liste von ausgesetzten Anfragen 164—168, die in dem globalen Speicher 32 gespeichert wird. Wie es z. B. in Fig. 4 dargestellt ist, kann die gespeicherte, ausgesetzte Anfrage 164 einen Prozessoridentifizierer 170, einen Abfragetypidentifizierer 172 und eine Blockadresse 174 umfassen. Andere Formate zum Speichern ausgesetzter Anfragen 164—168 können ebenfalls verwendet werden. Wenn die Aussetzungs-Flag 160 später auf den Nicht-Aussetzungs-Wert zurückgesetzt wird, nachdem die Teilwort-Datenverbindung vollendet ist, arbeitet die globale Speichersteuerung 30 die ausgesetzten Anfragen 164—168, die in der Warteschlange gespeichert sind, bevorzugterweise in einer Reihenfolge ab, bei der die zuerst empfangene Abfrage zuerst verarbeitet wird (d. h. in der empfangenen Reihenfolge).

Um festzustellen, wann das Verbinden beendet ist, ist es notwendig, nachzuverfolgen, wieviele Kopien des Datenblockes 118 zwischengespeichert wurden. Bevorzugterweise speichert die globale Speichersteuerung 30 einen Zählstand 178 für jeden Datenblock 118 in dem globalen Speicher 32, um diese Zahl nachzuverfolgen. Anfänglich stellt die globale Speichersteuerung 30 jeden Zählstand auf Null ein. Dann erhöht, jedesmal, wenn die globale Speichersteuerung 30 eine Kopie des Datenblocks 118 an einen Prozessor übermittelt, die Steuerung den Zählstand 178. Jedesmal wenn eine Kopie wieder mit dem Block 118 verbunden wird, erniedrigt die globale Speichersteuerung 30 den Zählstand 178, der dem Block zugeordnet ist. Auch wenn eine zwischengespeicherte Kopie aus dem lokalen Speicher eines Prozessors gelöscht wird, ohne daß diese verändert wurde, benachrichtigt die lokale Speichersteuerung des Prozessors die globale Speichersteuerung, so daß diese den Zählstand 178 erniedrigen kann. Auf diese Art wird der Zählstand 178 der Anzahl der ausstehenden zwischengespeicherten Kopien eines Datenblocks genau erhalten. Wenn der Zählstand 178 zurück auf Null erniedrigt ist, stellt die globale Speichersteuerung 30 fest, daß das Verbinden beendet ist und arbeitet die ausgesetzten Datenanfragen für diesen Datenblock ab, sofern welche vorhanden sind.

In den Fig. 1, 3 und 4 werden in einem vierten Ausführungsbeispiel der Erfindung die Aussetzungs-Flag 160, die Aussetzungs-Warteschlange 162 und der Zählstand 178 ebenfalls als eine "Sperre" verwendet, die es einem

Prozessor ermöglicht, exklusiven Zugriff auf einen Datenblock zu erhalten. Solche Sperren sind für synchronisierte Speicherzugriffe und für andere Zwecke sinnvoll. Um einen exklusiven Zugriff auf den Datenblock **118** im globalen Speicher zu erhalten, überträgt ein sperrender Prozessor (z. B. der Prozessor 22) eine Sperrabfrage an die globale Speichereinheit 26, in der der Datenblock **118** gespeichert ist. Wenn der Zählstand **178** für den Block **118** Null ist, setzt die globale Speichersteuerung 30 in der globalen Speichereinheit 26 die Aussetzungs-Flag 116, (die ebenfalls als "Sperr"-Flag dient. Für den Block auf einen gesperrten Wert,) kann eine Kopie des Datenblocks an den sperrenden Prozessor 22 zur Zwischenspeicherung übertragen und erhöht den Zählstand um Eins. Bei Ausführungsbeispielen, die eine Ein-Bit-Aussetzungs/Sperr-Flag **160** haben, wie sie z. B. in **Fig. 4** dargestellt ist, kann der gesperrte Wert Eins sein und ein anfänglicher ungesperrter Wert kann Null sein. Das Setzen der Aussetzungs/Sperr-Flag **160** verhindert, daß alle anderen Prozessoren eine Kopie des Datenblocks **118** zwischenspeichern, bis der Prozessor 22 die Sperre aufhebt. Daten- und Sperr-Anfragen für den Block **118** durch andere Prozessoren 23, 24, sind in der Aussetzungswarteschlange **162** angeordnet, während die Aussetzungs/Sperr-Flag **160** auf den gesperrten Wert gesetzt ist.

Wenn jedoch der Zählstand **178** von dem Block **118** nicht Null ist, wenn der sperrende Transistor **220** die Sperrabfrage überträgt, hat bereits ein anderer Prozessor eine zwischengespeicherte Kopie des Datenblocks. Deshalb kann dem sperrenden Prozessor 22 kein exklusiver Zugriff auf den Datenblock **118** gewährt werden, bis alle ausstehenden zwischengespeicherten Kopien mit dem Datenblock verbunden wurden, wodurch der Zählstand **178** auf Null zurückgeht. Bevorzugterweise reagiert unter solchen Umständen die globale Speichersteuerung 30 auf die Sperrabfrage durch Setzen der Aussetzungs/Sperr-Flag **160** und Speichern der Sperr-Anfrage an erster Stelle in der Aussetzungswarteschlange **162**. Dies verhindert ein weiteres Zwischenspeichern von Kopien des Blocks **118** und minimiert die Verzögerung beim Abarbeiten der Sperrabfrage. Sobald die ausstehenden zwischengespeicherten Kopien des Blocks **118** verbunden wurden, geht der Zählstand **178** auf Null und die Sperrabfrage kann abgearbeitet werden.

Das Aufheben der Sperre tritt auf, wenn der sperrende Prozessor 22 die Sperre durch Benachrichtigen der globalen Speichereinheit 26 ausdrücklich aufhebt. Als Reaktion setzt die globale Speichersteuerung 30 die Aussetzungs/Sperr-Flag **160** auf den ungesperrten Wert zurück.

Während der Datenblock **118** gesperrt ist, werden jegliche Datenabfragen durch andere Prozessoren für den Block ausgesetzt, d. h. in der Aussetzungswarteschlange zur späteren Abarbeitung gespeichert. Sobald die Sperre aufgehoben ist, werden die Anfragen durch die globale Speichersteuerung in der Reihe nach abgearbeitet. Durch Aussetzen von Datenanfragen arbeitet die Sperre auf ähnliche Weise, wie beim Zugreifen auf einen langsamen Speicher. Der Prozessor überträgt eine Datenabfrage an den Speicher und die angefragten Daten werden zurückgegeben, sobald diese verfügbar sind.

Das Aussetzen von Datenabfragen, während ein Datenblock gesperrt ist, hat ebenfalls den Vorteil der Vermeidung von "Spin-Sperren", die Programmierer bei anderen Arten von Sperren verwenden. Spin-Sperren sind eine programmierte Prozedur in einem Prozeß, der kontinuierlich den Status einer Sperre testet, bis die Sperre aufgehoben wird, und der Prozeß fortfahren kann. Spin-Sperren werden herkömmlicherweise bei Prozessen verwendet, die ohne den Zugriff auf bestimmte Daten, die gesperrt sein können, nicht weiter arbeiten können. Dieses kontinuierliche Testen eines Zustands der Sperre erzeugt jedoch einen erheblichen Kommunikationsverkehr zwischen einem Prozessor, der den Prozeß ausführt, und dem globalen Speicher und hält ebenfalls die zentrale Verarbeitungseinheit des Prozessors an. Die gleiche Funktionalität wie eine Spin-Sperre wird ohne den Kommunikationsverkehr und den Mehraufwand bei der zentralen Verarbeitungseinheit durch die Aussetzung von Datenabfragen unter Verwendung der Aussetzungswarteschlange bei der Sperre gemäß der vorliegenden Erfindung geschaffen.

Weiterhin können mit dieser Sperre die Prozessoren 22—24 mit ausgesetzten Datenabfragen daran gehindert werden, leerzulaufen, bis die Sperre durch die globalen Speichersteuerungen 26, 27, die den Prozessor benachrichtigen, wenn sie dessen Datenabfragen aussetzen, aufgehoben ist. In einigen Fällen können Prozesse einige Ziele mit geringerer Priorität haben, die während des Wartens auf die Aufhebung der gesperrten Daten, die für ein Ziel mit höherer Priorität erforderlich sind, durchgeführt werden. Durch Benachrichtigung des Prozessors von der Aussetzung seiner Datenabfrage kann der Prozeß, der auf dem Prozessor abläuft, andere sinnvolle Arbeit durchführen, die nicht die angeforderten Daten erfordert.

Wiederum ein weiterer Vorteil der Sperre gemäß diesem Ausführungsbeispiel der vorliegenden Erfindung besteht darin, daß, nachdem alle Funktionen der Sperre durch die globalen Speichereinheiten 26, 27 durchgeführt werden, keine Veränderungen der Prozessoren 22—24 durchgeführt werden müssen.

Bei einem fünften Ausführungsbeispiel der Erfindung, siehe **Fig. 1** und 5, werden eine globale Schreibe-Bitmaske **180** und eine globale Lese-Bitmaske **182** für jeden Datenblock **184** in dem globalen Speicher 32 (**Fig. 1**) mit dem Datenblock zur Identifikation von Datenwettlaufzuständen gespeichert. (Nur aus Gründen der Darstellung ist die Kopie **224** hier als durch den Prozessor 22 und der Block **196** als durch die globale Speichereinheit 26 beschrieben. Es ist offensichtlich, daß andere Prozessoren 23—24 und globale Speichereinheiten 27 des Computersystems **20** bevorzugterweise auf dieselbe Art und Weise arbeiten). Die globale Schreibe-Bitmaske **180** wird zur Nachverfolgung verwendet, welche Elemente **190—194** des Blocks **184** während dem Verbinden verändert sind und besteht aus einer Mehrzahl von Flags **200—204**, die in einer 1 : 1-Beziehung den Elementen zugeordnet sind. Die globale Lese-Bitmaske **182** wird zur Nachverfolgung verwendet, welche der Elemente **190—194** gelesen sind und besteht auf ähnliche Weise aus einer Mehrzahl von Flags **210—214**, die in einer 1 : 1-Beziehung den Elementen zugeordnet sind. Die Flags **200—204**, **210—214** sind anfänglich auf einen Nicht-geschrieben- bzw. Nicht-Gelesen-Wert gesetzt und werden auf diese Werte zurückgesetzt, wenn das Verbinden beendet ist. Bevorzugterweise ist jede Flag ein einzelnes Bit, wobei Null der Nicht-Geschrieben- und Nicht-Gelesen-Wert ist.

Bei dem fünften Ausführungsbeispiel der Erfindung werden ebenfalls eine lokale Schreibe-Bitmaske **220** und eine lokale Lese-Bitmaske **222** in dem lokalen Speicher 44 des Prozessors zusammen mit einer Kopie **224** des Datenblocks **184** gespeichert, wenn die Kopie durch den Prozessor 22 zwischengespeichert wird. Die lokale

Speichersteuerung 40 verfolgt, während die Kopie 224 in dem lokalen Speicher 44 zwischengespeichert ist, mit der lokalen Schreibe-Bitmaske 220, ob Datenelemente 230—234 der Kopie 224 verändert sind. Die lokale Schreibe-Bitmaske 220 besteht aus Flags 240—244, eine für jedes Element 230—234 der Kopie 224. Die lokale Speichersteuerung 40 setzt die Flags 240—244 anfänglich auf einen Nicht-Geschrieben-Wert, der anzeigt, daß die Elemente 230—234 bis jetzt nicht verändert wurden. Die lokale Lese-Bitmaske 222 hat ebenfalls Flags 250—254, die anfänglich auf einen Nicht-Gelesen-Wert gesetzt sind, eine für jedes der Elemente 230—234. Bevorzugterweise ist jede Flag ein einzelnes Bit, wobei Null der Nicht-geschrieben- und Nicht-Gelesen-Wert ist.

Wenn die zentrale Verarbeitungseinheit 36 die Elemente 230—234 der zwischengespeicherten Kopie 224 schreibt, setzt die lokale Speichersteuerung 40 die lokalen Schreibe-Bitmasken-Flags 240—244, die den geschriebenen Elementen zugeordnet sind, auf einen Geschrieben-Wert. Der Geschrieben-Wert ist wiederum bevorzugterweise Eins für eine Einzelbit-Flag und der Nicht-Geschrieben-Wert ist Null. Wenn die Elemente 230—234 der zwischengespeicherten Kopie 224 durch die zentrale Verarbeitungseinheit gelesen werden, setzt die lokale Speichersteuerung 40 die lokalen Lese-Bitmasken-Flags 250—254, die diesem gelesenen Element zugeordnet sind, auf einen Gelesen-Wert (bevorzugterweise auch Eins für eine Einzelbit-Flag). Folglich zeigen die lokale Schreibe- und Lese-Bitmaske 220, 222 an, welches Element der zwischengespeicherten Kopie 224 geschrieben bzw. gelesen wurde.

Nachdem die zwischengespeicherte Kopie 224 aus dem lokalen Speicher 44 gelöscht ist, überträgt die lokale Speichersteuerung 40 die lokalen Bitmasken 220, 222 und die zwischengespeicherte Kopie 224 an die globale Speichereinheit 26 zum Verbinden mit dem Datenblock 196. Die globale Speichersteuerung 30 verbindet die zwischengespeicherte Kopie 224 und den Datenblock 196 unter Verwendung der lokalen Schreibe-Bitmaske 220, wie es oben in Verbindung mit Fig. 2 beschrieben wurde.

Die lokale Schreibe- und Lese-Bitmaske 220, 222 werden ebenfalls in der globalen Speichereinheit 26 verwendet, um zu bestimmen, ob ein Datenwettlauf erzeugt wurde. Wie oben beschrieben wurde, ergibt sich ein Schreibe/Schreibe-Datenwettlauf, wenn mehr als ein Prozessor dasselbe Datenelement eines Datenblocks zwischen Synchronisationen schreibt. Um zu bestimmen, ob ein Schreibe/Schreibe-Datenwettlauf erzeugt wurde, werden die lokale Schreibe-Bitmaske 220 und die globale Schreibe-Bitmaske 180 verglichen, um zu bestimmen, ob Flags an entsprechenden Stellen der Bitmaske beide auf den Geschrieben-Wert gesetzt sind. Die globale Schreibe-Bitmasken-Flags 200—204 zeigen an, welche Elemente 190—194 des Blocks 196 bereits durch Verbinden anderer zwischengespeicherter Kopien mit dem Block verändert wurden. Die lokalen Schreibe-Bitmasken-Flags 240—244 zeigen an, welche Elemente 230—234 der Kopie 224 verändert wurden. Wenn die Flags an entsprechenden Stellen der Schreibe-Bitmasken 180, 220 folglich beide auf den Geschrieben-Wert gesetzt sind, wurde das gleiche Datenelement in der zwischengespeicherten Kopie, die derzeitig verbunden wird, und dem Datenblock durch eine vorhergehende verbundene Kopie verändert.

Die Bestimmung, daß Flags an entsprechenden Stellen der Schreibe-Bitmasken 118, 222 beide auf den Geschrieben-Wert gesetzt sind, kann schnell durchgeführt werden, wenn die Flags Einzelbits sind, der Geschrieben-Wert Eins ist und der nicht-Geschrieben-Wert Null ist. Dies wird durch Durchführen einer bitweisen logischen UND-Verknüpfung mit den Bitmasken 180, 220 durchgeführt. Wenn irgendwelche der Bits in dem Ergebnis der UND-Operation nicht Null sind, dann wurde sowohl in der zwischengespeicherten Kopie als auch in dem Datenblock ein gleiches Datenelement verändert. Die Stelle eines solchen nicht-Nullbits bei dem Ergebnis der UND-Operation zeigt weiterhin an, welche Datenelement oder -elemente sowohl in der zwischengespeicherten Kopie als auch in dem Datenblock verändert wurden. Diese Information kann zum Lokalisieren von Fehlern beim Prozeßablauf auf den Prozessoren 22, die Schreibe/Schreibe-Datenwettläufe verursachen, nützlich sein.

Weiterhin treten, wie oben beschrieben wurde, Lese/Schreibe-Datenwettläufe auf, wenn ein Prozessor ein Datenelement, das durch einen anderen Prozessor verändert wurde, zwischen Synchronisationen liest. Um zu bestimmen, ob ein Lese/Schreibe-Datenwettlauf erzeugt wurde, werden die lokale Lese-Bitmaske und die globale Schreibe-Bitmaske 180 verglichen. Die lokale Schreibe-Bitmaske 220 und die globale Lese-Bitmaske 182 werden verglichen. Wenn die Flags an entsprechenden Stellen der lokalen Lese- und globalen Schreibe-Bitmaske 222 und 180 auf den Gelesen- bzw. Geschrieben-Wert gesetzt sind, dann wurde ein gleiches Datenelement in der zwischengespeicherten Kopie 224, die derzeitig verbunden wird, gelesen und in dem Datenblock 196 durch eine vorher verbundene Kopie verändert. Wenn die Flags an entsprechenden Stellen der lokalen Schreibe- und globalen Lese-Bitmaske 220 und 182 ebenfalls auf den Geschrieben- bzw. Gelesen-Wert gesetzt sind, dann wurde ein gleiches Datenelement in der zwischengespeicherten Kopie 224, die derzeitig verbunden wird, verändert und in eine vorher verbundene Kopie des Datenblocks 196 gelesen. Wie beim Vergleich der lokalen und globalen Schreibe-Bitmasken 220, 180 werden diese Bestimmungen bevorzugterweise durch Anwendung einer bitweisen logischen UND-Operation auf die Bitmasken durchgeführt.

Nachdem die zwischengespeicherte Kopie 224 mit dem Datenblock 196 verbunden ist, wird die Schreibe-Bitmaske 220 mit der globalen Schreibe-Bitmaske 180 verbunden. Für jede der Flags 240—244 der Schreibe-Bitmaske 220 in der gespeicherten Kopie, die auf den Geschrieben-Wert eingestellt sind, stellt die globale Speichersteuerung 30 die globale Schreibe-Bitmasken-Flags 200—204 an einer entsprechenden Stelle auf den Geschrieben-Wert. Globale Schreibe-Bitmasken-Flags 200—204, die bereits auf den Geschrieben-Wert gesetzt sind, bleiben ebenfalls auf dem Geschrieben-Wert. Die Verbindung der Schreibe-Bitmasken kann durch eine bitweise logische ODER-Operation schnell durchgeführt werden, bei der die Schreibe-Bitmasken-Flags 200—204, 240—244 Einzelbitwerte sind, der Geschrieben-Wert Eins ist und der Nicht-Geschrieben-Wert Null ist. Die globale Speichersteuerung führt eine bitweise logische ODER-Operation bzgl. der Schreibe-Bitmasken 180, 220 durch und speichert die Ergebnisse in die globale Schreibe-Bitmaske 180.

Die Lese-Bitmaske 222 wird ebenfalls mit der globalen Lese-Bitmaske 182 auf eine ähnliche Art und Weise verbunden. Für jede der Flags 250—254 in der Lese-Bitmaske 222 der zwischengespeicherten Kopie setzt die globale Speichersteuerung 30 die globale Lese-Bitmas-

ken-Flag 210—214 an einer entsprechenden Stelle auf den Gelesen-Wert. Die globalen Lese-Bitmasken-Flags 210—214, die bereits auf den Gelesen-Wert gesetzt sind, bleiben auf dem Gelesen-Wert gesetzt. Bevorzugterweise hat jede Bitmaske Einzelbit-Flags, der Gelesen-Wert ist Eins und der Nicht-Gelesen-Wert ist Null, wobei die Verbindung der Bitmasken 182, 222 mittels der bitweisen logischen ODER-Operation ermöglicht wird.

Wenn die zwischengespeicherte Kopie 229 und der Datenblock 196 wie oben in Verbindung mit Fig. 2 beschrieben wurde, verbunden wird, können die Flags an entsprechenden Stellen der logischen und globalen Bitmasken 220, 222, 180, 182 beide als ein Ergebnis eines Lesevorgangs oder einer Veränderung des zugeordneten Datenelements in zwischengespeicherten Kopien des Blocks 196, die mehrere Male erneut zwischengespeichert wurden, durch einen einzelnen Prozessors auf den Gelesen- oder Geschrieben-Wert gesetzt sein. Dementsprechend identifizieren die Bitmasken sowohl das Lesen/Verändern der Datenelemente 190—194 durch mehrere Prozessoren, was einen Datenwettlauf erzeugt, und durch einen einzelnen Prozessor bei erneut zwischengespeicherten Kopien, was keinen Datenwettlauf erzeugt. Alternativ können die Prozessoren 22—24 daran gehindert werden, weitere Kopien des Datenblocks 196 nach dem Beginn der Verbindung unter Verwendung der Aussetzungs-Flag 160 (Fig. 4) und der oben beschriebenen Aussetzungswarteschlange 162 zwischenzuspeichern, so daß die Bitmasken lediglich tatsächliche Datenwettläufe identifizieren. Diese Alternative vermeidet das Verändern oder Lesen und Verändern eines Datenelements durch einen einzelnen Prozessor bei mehreren, erneut zwischengespeicherten Kopien des Datenblocks 196 durch Aussetzen weiterer Anfragen nach dem Datenblock, nachdem das Verbinden beginnt. Das Aussetzen der Anfrage erzeugt jedoch eine willkürliche Verzögerung bei diesen Prozessoren, bis alle ausstehenden Kopien des Blocks verbunden sind.

Bei einem sechsten Ausführungsbeispiel der Erfindung (siehe Fig. 1) wird der Kommunikationsverkehr zwischen den Prozessoren 22—24 und den globalen Speichereinheiten 26, 27 durch Durchführen bestimmter "Datenreduktions"-Operation in den globalen Speichereinheiten verringert (solche Operationen werden hier als globale Reduktions-Operationen bezeichnet). Globale Reduktions-Operationen können verschiedene Formen annehmen und können sowohl auf Datenwerte als auch Sperrwerte angewendet werden. Bei vielen Prozessen, die in dem Multiprozessor-Computersystem 20 ausgeführt werden, ist es wünschenswert, eine erste Form der globalen Reduktions-Operation bzgl. eines Datensatzes durchzuführen, die zu einem einzelnen Wert führt, der dann an einer bestimmten Stelle in einem Datenblock gespeichert ist. Ein Prozeß kann z. B. vorsehen, den Maximal- oder Minimal-Wert eines Arrays von Werten in einem Element eines Datenblocks zu speichern. Andere herkömmliche globale Reduktions-Operationen dieser Art schließen Speichern der Summe oder des Produkts eines Satzes von Datenelementen oder Speichern des Ergebnisses einer UND- oder ODER-Operation ein, die auf alle Elemente in einem Satz an einer bestimmten Stelle eines Datenblocks durchgeführt wird.

Gemäß dem sechsten Ausführungsbeispiel der Erfindung werden bestimmte Datenreduktions-Operationen durch die globalen Speichersteuerungen 30, 31 als Reaktion auf Datenreduktionsabfragen durch die Prozessoren 22—24 durchgeführt. Um z. B. eine Datenreduktions-Operation durchzuführen, die den Maximal-Wert eines Satzes von Datenelementen in dem globalen Speicher 32 an einer bestimmten Stelle speichert, übermittelt der Prozessor 22 eine Anfrage an die globale Speichersteuerung 30, in der er die "reduziere auf Maximum"-Operation, den Objektsatz an Datenelementen und den Ort, an dem das Ergebnis gespeichert werden soll, bestimmt. (Der bestimmte Satz von Datenelementen kann irgendwo in dem globalen Speicher 32 gespeichert werden). Als Reaktion führt die Steuerung 30 die angeforderte Operation aus. Die Steuerung 30 erfaßt den Maximal-Wert des bestimmten Satzes und speichert diesen Wert an der bestimmten Stelle.

Bei einer zweiten Form einer globalen Reduktions-Operation stellt ein Prozessor (z. B. der Prozessor 22), der die globale Reduktions-Operation anfordert, einen Operanden bereit und bestimmt eine globale Speicherstelle eines zweiten Operanden. Der abfragende Prozessor 22 bestimmt ebenfalls die bezüglich dieser Operanden auszuführende Operation. Die Operation selbst wird wiederum durch die globale Speichersteuerung 30 für die bestimmte Speicherstelle durchgeführt. Die globale Speichersteuerung 30 führt die Operation bezüglich der Operanden aus und speichert das Ergebnis in dem globalen Speicher 32, entweder anstelle des zweiten Operanden oder an einer anderen Stelle, die durch den Prozessor 22 bestimmt ist.

Das Durchführen von Datenreduktions-Operationen in den globalen Speichereinheiten 26, 27 reduziert den Kommunikationsverkehr zwischen den Prozessoren 22—24 und den globalen Speichereinheiten. Wenn Datenreduktions-Operationen durch die Prozessoren 22—24 selbst durchgeführt werden, müssen erhebliche Daten ausgetauscht werden. Die Prozessoren, die die Datenreduktions-Operation durchführen, müssen zuerst Datenabfragen an die globale Speichereinheit für den Objektsatz von Datenelementen übertragen. Als Reaktion überträgt die globale Speichereinheit Kopien des Datenblocks oder der Blöcke, die den Objektsatz enthalten an den anfordernden Prozessor zur Zwischenspeicherung. Der Prozessor führt die Datenreduktions-Operation bezüglich der zwischengespeicherten Kopien aus und schreibt das Ergebnis an die Zielstelle (die wiederum in einem anderen Datenblock sein kann und die Zwischenspeicherung eines solchen anderen Block erfordert). Der Prozessor überträgt dann die nun veränderte Kopie, die die Zielstelle enthält und die zugeordnete Schreibe-Bitmaske zurück an die globale Speichereinheit zum Verbinden mit dem Datenblock. Weiterhin erfordert jeder dieser verschiedenen Datenblockzugriffe im allgemeinen das Zugreifen auf eine Sperre, die diesem Datenblock zugeordnet ist (was das Warten auf die Beendigung eines Verbindungsprozesses notwendig machen kann), und hebt hinterher die Sperre auf.

Wenn die Datenreduktions-Operationen in den globalen Speichereinheiten 26, 27 durchgeführt werden, werden im Gegensatz dazu lediglich eine Datenreduktion und ein Datenwert zur Verwendung bei der Datenreduktionsabfrage (und in einigen Fällen ein Datenwert eines Operanden) von einem abfragenden Prozessor an eine globale Speichereinheit, die die Operation durchführt, übertragen. Auch der Mehraufwand, der dem Prozessor zugeordnet ist, der auf die Sperren der verschiedenen Datenblöcke zugreift, wird vermieden.

In Fig. 1 kann eine große Anzahl von Prozessoren 22—24 in dem Computersystem 20 zeitweise Zugriff auf einen einzelnen Datenblock erfordern, der in einer der globalen Speichereinheiten 26, 27 gespeichert ist. (Es ist

offensichtlich, daß die drei Prozessoren 22—24 und zwei globalen Speichereinheiten 26—27, die in Fig. 1 gezeigt sind, aus darstellerischen Gründen vorgesehen sind, und daß eine größere Anzahl von Prozessoren und globalen Speichereinheiten in dem Computersystem 20 eingeschlossen sein kann). In solchen Situationen muß die globale Speichereinheit, die den Datenblock speichert, eine große Anzahl von Datenabfragen abarbeiten und später die zwischengespeicherten Kopien des Blocks verbinden. Dementsprechend kann die Rate, bei der die globale Speichereinheit diese Abarbeitungs- und Verbindungs-Operationen durchführen kann, überschritten werden. Als ein Ergebnis werden einige der Abarbeitungs- und Verbindungs-Operationen verzögert, was einen sog. "Flaschenhals" erzeugt. Zu dieser Zeit können andere globale Speichereinheiten leerlaufen.

Bei dem siebten Ausführungsbeispiel der Erfindung wird dieser mögliche Flaschenhals, der aus einer großen Anzahl von Prozessoren resultiert, die auf einen einzelnen Datenblock zugreifen, der in einem Prozessor gespeichert ist, durch eine hierarchische gemeinsame Verwendung und Verbindung solcher Datenblöcke unter getrennten globalen Speichereinheiten vermieden. Wenn die globale Speichereinheit 26 z. B. beginnt, eine große Anzahl von Datenblockabfragen für den gleichen Datenblock zu empfangen (wenn z. B. die Anzahl von Datenblockabfragen eine vorbestimmte Grenze überschreitet), überträgt die globale Speichereinheit 26 eine Kopie des Datenblocks (im folgenden "ursprünglicher Datenblock" genannt) an eine weitere globale Speichereinheit. Beim Empfang der Kopie in der globalen Speichereinheit 27 speichert die globale Speichersteuerung 31 die Kopie (im folgenden "globale Kopie" genannt) in dem globalen Speicher 33. In Fig. 1 und 6 wird die Abarbeitung von weiteren Datenblockabfragen für den Block und die Verbindung von zwischengespeicherten Kopien des Blocks dann zwischen die globalen Speichereinheiten 26, 27 aufgeteilt. Bevorzugterweise wird die Aufteilung gemäß einer vorbestimmten hierarchischen Reihenfolge der Prozessoren und globalen Speichereinheiten durchgeführt. Eine solche hierarchische Reihenfolge 270 der Prozessoren 22—24 und globalen Speichereinheiten 26—27 ist in Fig. 6 gezeigt. Es ist offensichtlich, daß die hierarchische Reihenfolge eine logische Reihenfolge oder Zuordnung der Prozessoren zu den globalen Speichereinheiten ist, und keine Änderung der Struktur des Computersystems, die in Fig. 1 dargestellt ist, darstellt. Es ist ebenfalls offensichtlich, daß die hierarchischen Reihenfolgen gemäß diesem Ausführungsbeispiel der Erfindung viel mehr Prozessoren, globale Speichereinheiten und hierarchische Stufen einschließen können, als in Fig. 6 dargestellt ist. Insbesondere kann das Computersystem 20 zusätzliche globale Speichereinheiten einschließen, und globale Kopien der ursprünglichen Datenblöcke können in mehr als einem dieser globalen Speichereinheiten gespeichert sein. Auch werden viele Prozessoren typischerweise jedem der globalen Speichereinheiten, die eine globale Kopie des ursprünglichen Datenblocks speichern, zugeordnet sein.

Bei der bevorzugten hierarchischen Reihenfolge 270 sind einige der Prozessoren der globalen Speichereinheit 26 und andere der globalen Speichereinheit 27 zugeordnet. Solche Datenblockabfragen, die durch den Prozessor 22, der der globalen Speichereinheit 26 zugeordnet ist, übermittelt werden, werden durch diese Einheit abgearbeitet. Die Verbindung von Kopien des Blocks, der durch den Prozessor 22, der der Einheit 26 zugeordnet ist, zwischengespeichert ist, werden mit dem ursprünglichen Datenblock, wie es in Verbindung mit Fig. 2 beschrieben wurde, verbunden. Die Abarbeitung von Datenblockabfragen und die Verbindung von zwischengespeicherten Kopien für diese Prozessoren 22, 24, die der Einheit 27 zugeordnet sind, wird jedoch durch die Einheit 27 durchgeführt. In der globalen Speichereinheit 27 werden Kopien, die in den Prozessoren 23 und 24 zwischengespeichert sind, mit der globalen Kopie, die in dem globalen Speicher 33 gespeichert ist, verbunden. Auf ähnliche Weise wie das Verbinden, das oben in Verbindung mit Fig. 2 beschrieben wurde, wird jede zwischengespeicherte Kopie mit der globalen Kopie durch Speichern derjenigen Elemente der zwischengespeicherten Kopie, deren zugeordnete Flags in einer lokalen Schreibe-Bitmaske auf den Geschrieben-Wert gesetzt sind, in die globale Kopie verbunden. Eine globale Schreibe-Bitmaske wird in der Einheit 27 auch gespeichert, so daß die globale Kopie später mit dem ursprünglichen Datenblock, der in der Einheit 26 gespeichert ist, verbunden werden kann. Diese globale Schreibe-Bitmaske hat Flags, die in einer 1 : 1-Beziehung den Elementen der globalen Kopie zugeordnet sind. Diese Flags sind anfänglich auf den Nichtgeschrieben-Wert gesetzt. Wenn die Elemente der globalen Kopie während der Verbindung einer zwischengespeicherten Kopie verändert werden, werden die globalen Schreibe-Bitmasken-Flags, die solchen Elementen einer Flag zugeordnet sind, auf den Geschrieben-Wert gesetzt.

Sobald alle zwischengespeicherten Kopien der Prozessoren, die der Einheit 27 zugeordnet sind, mit der globalen Kopie verbunden sind, wird die globale Kopie mit dem ursprünglichen Datenblock verbunden. Um die globale Kopie und den ursprünglichen Datenblock zu verbinden, werden solche Elemente der globalen Kopie, deren zugeordnete globale Schreibe-Bitmasken-Flags auf den Geschrieben-Wert gesetzt sind, an entsprechenden Stellen des ursprünglichen Datenblocks gespeichert.

Bei einem achten Ausführungsbeispiel der Erfindung, das in Fig. 7 dargestellt ist, wird die Notwendigkeit, Anforderungen durch die Prozessoren 22—24 (Fig. 1) nach einem Datenblock 280, der in dem globalen Speicher 32 (Fig. 1) gespeichert ist, auszusetzen, sobald die Verbindung beginnt, durch ein alternatives Verbindungsschema vermieden, daß eine spezielle Verbindungs-Flag 282 und eine spezielle Verbindungstabelle 284, die dem Datenblock 280 zugeordnet ist, verwendet. Im Gegensatz zu dem ersten Ausführungsbeispiel der Erfindung, das oben beschrieben wurde, wird die Notwendigkeit, Anforderungen für teilweise verbundene Datenblöcke auszusetzen, mit einem Verbindungsschema, das lokale Schreibe-Bitmasken 56 (Fig. 2), die den zwischengespeicherten Kopien des Blocks zugeordnet sind, vermieden.

Das Verbindungsschema des achten Ausführungsbeispiels hat zwei Verbindungsmodi, einen "normalen" Modus und einen "speziellen" Modus, der durch den Zustand der speziellen Verbindungsflag 282 gesteuert ist. Die spezielle Verbindungsflag 282 hat zwei Zustände, einen ersten Zustand, der den normalen Verbindungsmodus anzeigt, und einen zweiten Zustand, der den speziellen Verbindungsmodus anzeigt. Anfänglich ist die spezielle Verbindungsflag 282 auf den ersten Zustand gesetzt und der normale Verbindungsmodus wird verwendet. Nachdem die Verbindung beginnt, und eine Anforderung durchgeführt wird, um eine Kopie des nun teilweise verbundenen Datenblocks 280 zwischenzuspeichern, wird der spezielle Verbindungsmodus ver-

wendet, und die spezielle Verbindungsflag 282 wird auf den zweiten Zustand gesetzt.

Bei dem anfänglichen, normalen Verbindungsmodus können Kopien des Datenblocks 280 durch die Prozessoren 22—24 angefordert und in ihren lokalen Speichern 44—46 (Fig. 1) gespeichert oder zwischengespeichert werden. Wenn die zwischengespeicherten Kopien aus den lokalen Speichern 44—46 gelöscht sind, werden sie mit dem Datenblock 280 in der globalen Speichereinheit 26 (Fig. 1) mit der Hilfe einer globalen Schreibe-Bitmaske 288, die dem Datenblock zugeordnet ist, verbunden. Die globale Schreibe-Bitmaske 288 enthält Bitmasken-Flags 290—293, um anzuzeigen, ob Datenelemente 296—299 des Datenblocks 280 seit der letzten vollständigen Verbindung verändert wurden. Anfänglich sind die Bitmasken-Flags 290—293 alle auf einen Nicht-Verändert-Wert gesetzt. Jede zwischengespeicherte Kopie wird durch Speichern dieser Datenelemente der zwischengespeicherten Kopie, die sich von entsprechend angeordneten, unveränderten Datenelementen 296—299 des Datenblocks unterscheidet, verbunden. Die Bitmasken-Flags 290—293 für die Datenblockelemente 296—299, die so verändert wurden, werden auf einen Verändert-Wert gesetzt.

Ein Zähler 304 verfolgt die Anzahl von zwischengespeicherten Kopien des Datenblocks 280, die noch ausstehen. Der Zähler 304 wird für jede Kopie des Datenblocks 280, die durch die Prozessoren 22—24 zwischengespeichert werden, erhöht und für jede zwischengespeicherte Kopie des Datenblocks 280, die aus den lokalen Speichern 40—46 gelöscht wird, erniedrigt. Die Verbindung ist vollständig, wenn der Zähler 304 auf Null zurückkehrt. Daraufhin werden alle globalen Schreibe-Bitmasken-Flags 290—293 auf den Nicht-Verändert-Wert zurückgesetzt.

Nachdem die Verbindung in dem normalen Verbindungsmodus beginnt (d. h. zumindest eine zwischengespeicherte Kopie wurde mit dem Datenblock 280 verbunden, wobei zumindest eines der Datenelemente 296—299 verändert wird), verursacht jegliche nachfolgende Anfrage nach einer Kopie des Datenblocks 280 durch die Prozessoren 22—24 den Eintritt in den speziellen Verbindungsmodus. (Bei dem bekannten Karp-Sarkar-Verbindungsschema würden solche nachfolgenden Anforderungen ausgesetzt). Um festzustellen, ob die Verbindung begonnen hat, kann die globale Schreibe-Bitmaske 280 getestet werden. Wenn irgendeine der Bitmasken-Flags 290—293 auf den Geändert-Wert gesetzt ist, hat die Verbindung begonnen. Wenn die globale Speichersteuerung 30 (Fig. 1) eine Anfrage nach einer Kopie des Datenblocks 280 empfängt und die globale Schreibe-Bitmaske ist nicht Null (wobei der Verändert-Wert der Bitmasken-Flags 1 Bit ist), setzt die Steuerung 30 dementsprechend die spezielle Verbindungsflag auf den zweiten Zustand, um in den speziellen Verbindungsmodus einzutreten.

In dem speziellen Verbindungsmodus reagiert die globale Speichersteuerung 30 auf eine Anfrage nach einer Kopie des Datenblockes 280 (einschl. der Anfrage, die den Eintritt in den speziellen Verbindungsmodus verursacht hat) durch einen der Prozessoren 22—24 durch Übertragen der Kopie an den abfragenden Prozessor. (Solche Kopien, die während des speziellen Verbindungsmodus erhalten werden, sind "teilweise verbundene" Kopien, da sie erhalten wurden nachdem die Verbindung des Datenblocks 280 begonnen hat). Die globale Speichersteuerung 30 verfolgt diese teilweise verbundenen Kopien unter Verwendung der speziellen Verbindungstabelle 284. Die globale Speichersteuerung speichert ein Duplikat der teilweise verbundenen Kopie 308 (identisch zu derjenigen, die an den abfragenden Prozessor gesendet wurde) in einem unbesetzten Schlitz 309 der speziellen Verbindungstabelle 284 zusammen mit einem Prozessoridentifizierer 310 (um den abfragenden Prozessor zu identifizieren) und einer Blockadresse 312 (um den Datenblock 280 zu identifizieren, von dem die Duplikatkopie herstammt). Die Duplikatkopie 308, der Prozessoridentifizierer 310 und die Blockadresse 312 bilden einen "Eintrag" in die spezielle Verbindungstabelle. Die Blockadresse 212 ermöglicht es, die spezielle Verbindungstabelle 284 für mehr als einen Datenblock in dem globalen Speicher zu verwenden. Alternativ können spezielle Verbindungstabellen (einer für jeden Datenblock) nach Bedarf aus einem unbenutzten Abschnitt des globalen Speichers 32 (Fig. 1) zugeordnet werden. Mit solchen alternativen speziellen Verbindungstabellen muß die Blockadresse 312 nicht gespeichert werden.

Beim Verbinden einer zwischengespeicherten Kopie mit Datenblock 280 im speziellen Verbindungsmodus sucht die globale Speichersteuerung die spezielle Verbindungstabelle 284 nach einem Eintrag ab, der den Prozessoridentifizierer 310 des Prozessors, der die Kopie zwischengespeichert hat, enthält, und die Blockadresse 312 für den Datenblock 280, von woher die zwischengespeicherte Kopie herstammt, um festzustellen, ob die zwischengespeicherte Kopie eine teilweise verbundene Kopie ist. Wenn die Tabelle keinen solchen Eintrag enthält (was anzeigt, daß die zwischengespeicherte Kopie keine teilweise verbundene Kopie ist), werden die zwischengespeicherte Kopie und der Datenblock 280 wie im normalen Verbindungsmodus unter Verwendung der globalen Bitmaske 288 verbunden. Wenn jedoch ein solcher übereinstimmender Eintrag (z. B. Eintrag 309) existiert (wodurch angezeigt ist, daß die zwischengespeicherte Kopie eine teilweise verbundene Kopie ist), wird die zwischengespeicherte Kopie unter Verwendung einer Aktualisierungs-Bitmaske 316 verbunden.

Beim Verbinden einer teilweise verbundenen Kopie erzeugt die globale Speichersteuerung 30 die Aktualisierungs-Bitmaske 316, die Bitmasken-Flags 320—323 enthält, die den Datenelementen der teilweise verbundenen Kopie (nicht gezeigt) zugeordnet sind. Die Steuerung setzt diese Bitmasken-Flags 320—323 auf den Geändert-Wert, die Datenelementen der teilweise verbundenen Kopie zugeordnet sind, die sich von entsprechend angeordneten Datenelementen der Duplikatkopie 308 bei dem übereinstimmenden Eintrag 309 unterscheiden. Diese Aktualisierungs-Bitmaske 316 steuert die Verbindung. Die Steuerung 30 speichert solche Datenelemente der teilweise verbundenen Kopie, deren zugeordnete Bitmasken-Flags 320—323 auf den Geändert-Wert gesetzt sind, in dem globalen Datenblock 280. Der übereinstimmende Eintrag 309 in der Warteschlange wird dann gelöscht.

Nach dem Verbinden der teilweise verbundenen Kopie in den globalen Datenblock 380, verbindet die globale Speichersteuerung 30 ebenfalls die Aktualisierungs-Bitmaske 316 mit der globalen Bitmaske 288. Für jede der Bitmasken-Flags 320—323 der Aktualisierungs-Bitmaske 360, die auf den Geändert-Wert gesetzt ist, setzt die Steuerung 30 die entsprechend angeordneten Bitmasken-Flags 290—293 der globalen Bitmaske 288 auf den Geändert-Wert. Wenn der Geänder-Wert ein einzelner Ein-Bit und der Nicht-geändert-Wert ein einzelner Null-Bit ist, kann die Verbindung durch Speichern

des Ergebnisses einer bitweisen logischen ODER-Verknüpfung der globalen und der Aktualisierungs-Bitmaske **288**, **316** in der globalen Bitmaske **288** bewirkt werden.

Das gerade beschriebene, alternative Verbindungsschema des achten Ausführungsbeispiels der Erfindung kann ebenfalls auf das Teilwort-Verbindungs-Schema des dritten Ausführungsbeispiels, das oben beschrieben wurde, angewendet werden, um die Notwendigkeit zur Aussetzung von Prozessorabfragen auszuschließen, die nach dem Beginn der Teilwort-Verbindung auftreten. Bei einem solchen Ausführungsbeispiel der Erfindung, bei der ein Teilwort-Speicher-Teilungs-Schema das alternative Verbindungsschema ersetzt, werden Anfragen, die nach dem Verbinden einer zwischengespeicherten Kopie, die durch eine Teilwort-Speicher-Operation verändert wurde, mit einem Datenblock im globalen Speicher, nicht ausgesetzt. Statt dessen sendet die globale Speichersteuerung **30** (**Fig. 1**) die angeforderte Kopie des teilweise verbundenen Datenblocks an den abfragenden Prozessor, macht einen entsprechenden Eintrag in die spezielle Verbindungstabelle, und setzt eine spezielle Verbindungsflag. Danach werden verbundene Kopien, die übereinstimmende Tabelleneinträge haben, mit einer Aktualisierungs-Bitmaske verbunden, und zwischengespeicherte Kopien ohne übereinstimmende Einträge werden mit einer globalen Bitmaske wie beim achten Ausführungsbeispiel verbunden. Bevor die spezielle Verbindungsflag gesetzt wird, werden zwischengespeicherte Kopien wie beim dritten Ausführungsbeispiel verbunden.

In den **Fig. 1** und 3 schafft ein neuntes Ausführungsbeispiel der Erfindung eine Datenverbindung in den globalen Speichereinheiten **26**, **27** (**Fig. 1**) auf der Bit-Ebene. Mit dieser Fähigkeit kann mehr als einer der Prozessoren **22**—**24** (**Fig. 1**) Bits in demselben Byte eines gemeinsamen Datenblocks verändern, solange nicht zwei Prozessoren den gleichen Bit zwischen Synchronisationen verändern. Die getrennt veränderten Bits in der zwischengespeicherten Kopie jedes Prozessors werden später auf der Bit-Ebene zu dem ursprünglichen Datenblock im globalen Speicher verbunden. Jegliche Datenwettläufe, die durch gemeinsame Verwendung der Daten zwischen mehreren Prozessoren erzeugt werden, können auf der Bit-Ebene erfaßt werden.

Bei dem neunten Ausführungsbeispiel der Erfindung (wie bei dem dritten Ausführungsbeispiel, siehe **Fig. 3**) ist eine Teilwort-Speicher-Flag **110** und eine Schreibe-Bitmaske **88** der zwischengespeicherten Kopie jeder zwischengespeicherten Kopie **86**, die in den lokalen Speichern **44**—**46** der Prozessoren gespeichert ist, zugeordnet. Die Teilwort-Speicher-Flag **110** umfaßt in dem neunten Ausführungsbeispiel bevorzugterweise einen Zwei-Bit-Wert zum Anzeigen der Datenelemente mit minimaler Größe, die in der zwischengespeicherten Kopie **86** verändert wurden. Mit einem Zwei-Bit-Wert können vier minimale Größen angezeigt werden. Ein Wert von "11" kann z. B. verwendet werden, um anzuzeigen, daß einzelne Bit der zwischengespeicherten Kopie verändert wurden. Werte von "10", "01", und "00" können verwendet werden, um anzuzeigen, daß Bytes, Halbwörter (im allgemeinen 16 Bit-Datenelemente) bzw. Wörter die veränderten Datenelemente mit der minimalen Größe wahren.

Wenn bei dem neunten Ausführungsbeispiel ein Wort, Halbwort oder Byte die minimale Größe eines veränderten Datenelements ist, wird die zwischengespeicherte Kopie **86** mit ihrem ursprünglichen Datenblock auf dieselbe Art wie beim dritten Ausführungsbeispiel verbunden. Insbesondere wenn ein Wort das kleinste veränderte Datenwort ist, wird die zwischengespeicherte Kopie unter Verwendung der Schreibe-Bitmaske **88** der zwischengespeicherten Kopie verbunden. Wenn die minimale Größe des veränderten Datenelements ein Halbwort oder ein Byte ist, wird die zwischengespeicherte Kopie unter Verwendung einer Teilwort-Schreib-Bitmaske **124** (**Fig. 4**) verbunden, die eine Bitmasken-Flag für jedes Halbwort bzw. Byte des ursprünglichen Datenblocks **118** (**Fig. 4**) hat. Zwischengespeicherte Kopien, die auf der Bit-Ebene verändert sind, werden wie unten beschrieben wird, verbunden. Sobald der ursprüngliche Datenblock **118** auf der Halbwort-, Byte- oder Bit-Ebene verbunden ist, werden jegliche zurückbleibenden zwischengespeicherten Kopien des Datenblocks ebenfalls auf der gleichen Ebene verbunden, bis eine Verbindung auf niedrigerer Stufe notwendig ist oder die Verbindung abgeschlossen ist.

Für die meisten zwischengespeicherten Kopien ist das veränderte Element mit der geringsten Größe im allgemeinen das Wort. Der Verbindungsprozeß für solche zwischengespeicherte Kopien ist der schnellste, erfordert am wenigstens Verarbeitungsaufwand und eine kleinere Bitmaske. Zwischengespeicherte Kopien, deren veränderte Elemente mit minimaler Größe Bytes oder Halbwörter sind, treten weniger häufig auf. Die Verbindung benötigt dann einen größeren Verarbeitungsaufwand und mehr Zeit. Die Verbindung von zwischengespeicherten Kopien, die auf der Bit-Ebene verändert wurden, ist sogar noch weniger häufig und es wird noch mehr Verarbeitungszeit und -aufwand verbraucht. Folglich besteht der Vorteil der Verbindung auf der Ebene des veränderten Datenelements mit geringster Größe darin, daß die Verarbeitungszeit und -aufwand für eine gegebene zwischengespeicherte Kopie minimiert werden, während die gemeinsame Datenverwendung zwischen Prozessoren bis zur Bit-Ebene hinunter möglich wird, wenn dies notwendig ist.

Zum Verbinden auf der Bit-Ebene ist es notwendig, zu erfassen, wann Bit-Ebenen-Veränderungen der zwischengespeicherten Kopie **86** durchgeführt werden (siehe **Fig. 1**, 3 und 8).

Mit herkömmlichen CPUs **36**—**38** verändern im allgemeinen lediglich Speicher-Operationen (in Wort- oder Bit-Größeninkrementen), die durch die CPUs ausgeführt werden, direkt die Daten in zwischengespeicherten Kopien, die in lokalen Speichern **44**—**46** gespeichert sind. Veränderungen eines Worts oder eines Bytes in solchen zwischengespeicherten Kopien können deshalb erfaßt werden, wenn eine CPU **36**—**38** eine Speicher-Operation durchführt, um das Wort, Halbwort oder Byte von seinem inneren Register an die Lokalspeicher **44**—**46** zu bewegen. Nachdem lediglich Lade- und Speicher-Operationen Daten zwischen den CPUs **36**—**38** und lokalen Speichern **44**—**46** bewegen, werden einzelne Bits an einer zwischengespeicherten Kopie indirekt verändert. Um einzelne Bits zu verändern, laden die CPUs **36**—**38** ein inneres Register mit einem Byte oder Wort an Daten, führen eine oder mehrere Bit-Ebenen-Operationen bezüglich der Daten durch, speichern dann das Wort oder Byte in der zwischengespeicherten Kopie in den lokalen Speichern **44**—**46**.

Um Bit-Ebenen-Veränderungen zu erfassen, haben die CPUs **36**—**38** beim neunten Ausführungsbeispiel eine Bit-Verändert-Flag für jedes ihrer inneren Register. Wie es z. B. in **Fig. 8** dargestellt ist, kann die CPU **36** acht innere Register **330**—**337** zur vorübergehenden Spei-

cherung von Daten, die durch die funktionale Einheit 338 der CPU verarbeitet wird, umfassen. Im allgemeinen enthält die Funktionseinheit 338 Ganzzahl- und Gleit-Komma-Verarbeitungsschaltungen und eine Befehlsdekodierungsschaltung. Die CPU 36 hat für jedes innere Register 330—337 ebenfalls Bit-Verändert-Flags 340—347 mit einem Bit. Die CPU 36 überträgt Daten zwischen den inneren Registern 330—337 und dem lokalen Speicher 344 (Fig. 1) über einen Prozessor-Bus 350 und eine Bus-Schnittstelle 352.

Die Bit-Verändert-Flags 340—347 zeigen an, ob Daten in ihren zugeordneten Registern 330—337 auf der Bit-Ebene verändert wurden. Wenn die CPU Daten in irgendeines der inneren Register 330—337 lädt, ist die Bit-Verändert-Flag 340—347 für das Register anfänglich auf einen Unverändert-Wert (z. B. Null) gesetzt. Wenn die CPU 36 innerhalb ihrer Funktionseinheit 338 einen Befehl ausführt, der eines oder mehrere Bits der Daten in einem Register verändert, aber nicht ein gesamtes Byte oder Wort, setzt die Funktionseinheit 338 die Bit-Verändert-Flag auf einen Verändert-Wert (z. B. Eins). Wenn die CPU 36 die Daten aus einem der inneren Register 330—337 in der zwischengespeicherten Kopie 86 in dem lokalen Speicher 44 speichert, überprüft die lokale Speichersteuerung 40 die Bit-Verändert-Flag, die diesem Register zugeordnet ist, um zu bestimmen, ob die zwischengespeicherte Kopie auf der Bit-Ebene verbunden werden sollte, und setzt dementsprechend die Teilwort-Verändert-Flag 110 für die zwischengespeicherte Kopie.

Bei dem neunten Ausführungsbeispiel (vgl. Fig. 4) wird die zwischengespeicherte Kopie 86 mit ihrem ursprünglichen Datenblock 118 auf der Bit-Ebene verbunden, wenn die Teilwort-Speicher-Flag 110 auf "11" gesetzt ist, was anzeigt, daß die minimale Größe des veränderten Elements das Bit ist. Für eine Bit-Ebenen-Verbindung sind die Bitmasken-Flags 126—137 der Teilwort-Speicher-Bitmaske 124 in einer 1 : 1-Beziehung den Bits des Datenblocks 118 zugeordnet (Fig. 4 stellt den Fall dar, bei dem die Bitmasken-Flags 126—137 in einer 1 : 1-Beziehung den Bytes 146—157 des Datenblocks für eine Byte-Ebenen-Verbindung zugeordnet sind). Bevorzugterweise ist jede der Bitmasken-Flags 126—137 ein Einzel-Bit-Wert, wobei eine Eins anzeigt, daß das zugeordnete Bit des Datenblocks 118 verändert wurde (wie bei einer vorhergehenden Verbindung) und eine Null keine Veränderung anzeigt. Durch Verwendung dieser Teilwort-Speicher-Bitmaske 124 werden die veränderten Bits der Zwischengespeicherten Kopie 86 an Stellen, die den unveränderten Bits des Datenblocks 118 entsprechen, in dem Datenblock gespeichert, um die Bit-Ebenen-Verbindung zu bewirken.

Bei der bevorzugten Bit-Ebenen-Verbindung erzeugt die globale Speichersteuerung 30 (Fig. 1) für den ursprünglichen Datenblock 118 (unter der Annahme, der Datenblock 118 ist in dem globalen Speicher 32 gespeichert) eine vorübergehende Bitmaske (nicht gezeigt) mit einer Einzel-Bit-Flag für jedes Bit des Datenblocks 118 aus einer bitweisen EXCLUSIVE-ODER-Verknüpfung der zwischengespeicherten Kopie 86 des Blocks 118. Dies setzt jede vorübergehende Bitmasken-Flag auf Eins, die einem Bit-Datenblock 318, das sich von dem entsprechend angeordneten Bit in der zwischengespeicherten Kopie 86 unterscheidet, zugeordnet ist. Als nächstes speichert die globale Speichersteuerung 30 für jedes Bit des Datenblocks 118, das im vorhergehenden nicht verändert wurde, das entsprechend angeordnete Bit der zwischengespeicherten Kopie 86 anstelle dieses Bits. Dies kann durch eine bitweise logische UND- und ODER-Operation durchgeführt werden, wie es durch die folgende Gleichung dargestellt ist:

$$B = (B \text{ UND NICHT } M) \text{ ODER } (C \text{ UND } M) \quad (1).$$

Wobei B den Datenblock 118 darstellt, M stellt die Teilwort-Speicher-Bitmaske 124 dar, C stellt die zwischengespeicherte Kopie 86 dar, UND ist die logische UND-Operation, ODER ist die bitweise logische ODER-Operation, und NICHT ist die logische Komplementär-Operation. Die globale Speichersteuerung 30 führt dann die bitweise logische ODER-Operation der Teilwort-Speicher-Bitmaske 124 mit der vorübergehenden Bitmaske durch, um die Teilwort-Speicher-Bitmaske zu aktualisieren.

Nachdem die Halbwort-, Byte- und Bit-Ebenen-Verbindung gemäß den unveränderten Datenelementen des Datenblocks gesteuert sind, ist es notwendig, die Möglichkeit einzubeziehen, daß ein einzelner Prozessor einen Datenblock mehr als einmal zwischen Synchronisationen zwischenspeichert und verändert. Entweder kann das Aussetzungsschema des dritten Ausführungsbeispiels oder das alternative Verbindungsschema des achten Ausführungsbeispiels für diese Aufgabe verwendet werden.

Patentansprüche

1. Verfahren zur gemeinsamen Verwendung eines Speichers in einem Multiprozessor-Computersystem (20), mit folgenden Schritten:
Speichern einer Mehrzahl von Datenblöcken (118, 196, 280) in einem globalen Speicher (32);
für jeden anfragenden Prozessor (22—24), Speichern einer Kopie (54, 86, 224) eines ursprünglichen Datenblocks in einem Cache-Speicher (44—46), der dem abfragenden Prozessor örtlich zugeordnet ist, zum Zugriff durch den abfragenden Prozessor;
Nachverfolgen, welche Elemente (70—74, 80—82, 230—234) der Kopien durch ihren abfragenden Prozessor verändert werden; und
Verbinden der Kopien durch Speichern dieser Elemente der Kopien, die verändert wurden, in dem Datenblock.

2. Verfahren nach Anspruch 1, bei dem der Schritt des Nachverfolgens folgende Schritte aufweist:
Speichern einer Bitmaske (56, 88, 220) in dem Cache-Speicher (40—46), die eine Flag (60—64, 90—92, 240—244) umfaßt, die jedem Datenelement (70—74, 80—82, 230—234) der Kopie (54, 86, 224) zugeordnet ist, wobei jede Flag anfänglich auf einen ersten Wert gesetzt ist;
Setzen der Flag in der Bitmaske, die jedem der Datenelemente zugeordnet ist, die durch den abfragenden Prozessor geschrieben sind, auf einen zweiten Wert; und
wobei der Schritt des Verbindens der Kopie und des ursprünglichen Datenblocks das Speichern jedes Datenelements der Kopie dessen zugeordnete Flag auf den zweiten Wert gesetzt ist im ursprünglichen Datenblock (118, 196, 218) umfaßt.

3. Verfahren zur gemeinsamen Verwendung eines Speichers (32, 33) bei mehreren Prozessoren (22—24), das folgende Schritte aufweist:
Speichern gemeinsamer Daten in einem globalen Speicher (32, 33), wobei die gemeinsamen Daten eine Mehrzahl von Datenblöcken (118, 196, 280)

umfassen;
für jeden Prozessor (22—24), der auf Daten in einem der Datenblöcke zugreift:
Speichern einer lokalen Kopie (54, 86, 224) des Datenblocks in dem lokalen Speicher des Prozessors;
Zuordnen einer Mehrzahl von Schreibe-Flags (60—64, 90—92, 240—244) einer lokalen Kopie in dem lokalen Speicher in einer 1 : 1-Beziehung zu einer Mehrzahl von Datenelementen (70—74, 80—82, 230—234) mit einer ersten Größe in der lokalen Kopie, wobei jede der Schreibe-Flags der lokalen Kopie anfänglich auf einen Nicht-geschrieben-Wert gesetzt ist;
für jedes der Datenelemente mit der ersten Größe der lokalen Kopie, das durch den Prozessor verändert wird, Setzen seiner zugeordneten Schreibe-Flags der lokalen Kopie auf einen Geschrieben-Wert;
Übermitteln der lokalen Kopie und der Schreibe-Flags der lokalen Kopie an den globalen Speicher, wenn die lokale Kopie aus dem lokalen Speicher gelöscht wird;
Verbinden der lokalen Kopie und des Datenblocks in dem globalen Speicher, für jede der Schreibe-Flags der lokalen Kopie, die auf einen Geschrieben-Wert gesetzt sind, durch Speichern des Datenelements der ersten Größe, daß einer solchen Schreib-Flag der lokalen Kopie zugeordnet ist, in dem Datenblock; und
wenn mehrere Prozessoren gleichzeitig auf den Datenblock zugreifen:
Identifizieren des Auftretens einer Schreibe/Schreibe-Datenwettlaufbedingung, bei der zumindest zwei Prozessoren Schreibe-Flags der lokalen Kopie, die beide auf den Geschrieben-Wert gesetzt sind und Datenelementen mit einer ersten Größe zugeordnet sind, an entsprechende Stellen ihrer jeweiligen lokalen Kopien senden.

4. Verfahren nach Anspruch 3, bei dem der Schritt des Verbindens der lokalen Kopie (54, 86, 224) und des Datenblocks (118, 196, 288) in dem globalen Speicher (32, 33) folgende Schritte aufweist:
Zuordnen einer Mehrzahl von Datenblock-Schreibe-Flags (290—293) in dem globalen Speicher zu einer Mehrzahl von Datenelementen (296—299) mit einer ersten Größe in dem Datenblock, wobei jede der Datenblock-Schreibe-Flags anfänglich auf einen Nicht-geschrieben-Wert gesetzt ist;
Zuordnen einer speziellen Verbindungs-Flag (282) zu dem Datenblock in dem globalen Speicher, wobei die spezielle Verbindungs-Flag anfänglich auf einen ersten Wert gesetzt ist;
für jede gespeicherte lokale Kopie, während der Datenblock teilweise verbunden ist:
Speichern einer Duplikatkopie (308) des teilweise verbundenen Datenblocks in einem Tabelleneintrag (309) für die lokale Kopie; und
Setzen der speziellen Verbindungsflag auf einen zweiten Wert;
wenn die spezielle Verbindungsflag auf den ersten Wert gesetzt ist, Speichern solcher Datenelemente der lokalen Kopie, die sich von den entsprechend angeordneten Datenelementen der ersten Größe des Datenblocks unterscheiden, deren zugeordnete Datenblock-Schreibe-Flag auf den Nicht-geschrieben-Wert in dem Datenblock gesetzt ist;
wenn die spezielle Verbindungsflag auf den zweiten Wert gesetzt ist und die lokale Kopie gespeichert wurde, als der Datenblock nicht teilweise verbunden war, Speichern solcher Datenelemente der lokalen Kopie in dem Datenblock, die sich von den entsprechend angeordneten Datenelementen der ersten Größe des Datenblocks unterscheiden, während zugeordnete Datenblock-Schreibe-Flag auf den Nicht-geschrieben-Wert gesetzt ist; und
wenn die spezielle Verbindungsflag auf den zweiten Wert gesetzt ist und die lokale Kopie gespeichert wurde, als der Datenblock teilweise verbunden war, Speichern derjenigen Datenelemente der lokalen Kopie in den Datenblock, die sich von entsprechend angeordneten Elementen der Duplikatkopie in dem Tabelleneintrag für die lokale Kopie unterscheiden.

5. Verfahren nach Anspruch 3, das ferner folgende Schritte aufweist:
für jeden Prozessor (22—24), der auf Daten in einem der Datenblöcke zugreift:
Zuordnen einer Teilwort-Speicher-Flag (110) zu der lokalen Kopie (86) in dem lokalen Speicher (44—46) des Prozessors, wobei die Teilwort-Speicher-Flag die minimale Größe eines veränderten Datenelements anzeigt und anfänglich gesetzt ist, um eine erste Größe anzuzeigen, wobei die erste Größe diejenige der Datenelemente mit der ersten Größe (80—82) ist;
Setzen der Teilwort-Speicher-Flag, um eine zweite Größe anzuzeigen, wenn der Prozessor eine Teilwort-Speicher-Operation bezüglich der lokalen Kopie durchführt; und
Senden der Teilwort-Speicher-Flag zusammen mit der lokalen Kopie und den lokalen Kopie-Schreibe-Bit- an den globalen Speicher, wenn die lokale Kopie aus dem lokalen Speicher gelöscht wird;
wenn die Teilwort-Speicher-Flag eingestellt ist, um die zweite Größe anzuzeigen;
Zuordnen einer Mehrzahl von Datenblock-Schreibe-Flags (126—137) in dem globalen Speicher (32, 33) 1 : 1 zu einer Mehrzahl von Datenelementen (146—157) der zweiten Größe in dem Datenblock (118), wobei die zweiten Datenelemente die zweite Größe haben, wobei die Datenblock-Schreibe-Flags anfänglich auf den Nicht-geschrieben-Wert gesetzt sind; und
für jedes Datenelement der zweiten Größe des Datenblocks, dessen zugeordnete Flag auf den Nicht-geschrieben-Wert gesetzt ist, wenn sich ein Datenelement (96—107) an einer entsprechenden Stelle der lokalen Kopie und ein solches Datenelement mit zweiter Größe unterscheiden, Speichern der lokalen Kopie des Datenelements anstelle eines solchen Datenelements mit zweiter Größe und Setzen der Datenblock-Schreibe-Flag auf dem Geschrieben-Wert, die dem Datenelement mit zweiter Größe zugeordnet ist; und
wenn die Teilwort-Speicher-Flag gesetzt ist, um die erste Größe anzuzeigen:
für jede der Schreibe-Flags der lokalen Kopie, die auf den Geschrieben-Wert gesetzt sind, Speichern des Datenelements mit der ersten Größe, das einer solchen Schreibe-Flag der lokalen Kopie zugeordnet ist, in den Datenblock.

6. Verfahren nach Anspruch 3, das ferner folgende Schritte aufweist:
für jeden Prozessor (22—24), der auf Daten in einem der Datenblöcke (196) zugreift:
Zuordnen einer Mehrzahl von Lese-Flags

(250—253) in dem lokalen Speicher (224) in einer 1 : 1-Beziehung mit den Datenelementen mit der ersten Größe (230—234) der lokalen Kopie (224), wobei die Lese-Flags anfänglich auf einen Nicht-gelesen-Wert gesetzt sind;
wenn ein Datenelement mit erster Größe durch den Prozessor gelesen wird, Setzen seiner zugeordneten Lese-Flag auf einen Gelesen-Wert; und
Übermitteln der Lese-Flag zusammen mit der lokalen Kopie und den Schreibe-Flags (240—244) der lokalen Kopie an den globalen Speicher (32, 33), wenn die lokale Kopie aus dem lokalen Speicher gelöscht wird; und
wenn mehrere Prozessoren (22—24) gleichzeitig auf den gleichen Datenblock (196) zugreifen:
Identifizieren des Auftretens eines Lese-Schreibe-Datenwettlaufs, wobei ein Prozessor eine Schreibe-Flag der lokalen Kopie, die auf den Geschrieben-Wert gesetzt ist, sendet, und ein weiterer Prozessor eine Lese-Flag, die auf den Gelesen-Wert gesetzt ist, sendet, und wobei die Schreibe- und Lese-Flags den Datenelementen mit der ersten Größe an den entsprechenden Stellen ihren jeweiligen lokalen Kopien zugeordnet sind.

7. Verfahren nach Anspruch 3, das ferner folgende Schritte aufweist:
Zuordnen einer Sperr-Flag (160) zu jedem Datenblock (118) in dem globalen Speicher (32, 33), wobei die Sperr-Flag anfänglich auf einen Ungesperrt-Wert gesetzt ist;
Setzen der Sperr-Flag auf einen Gesperrt-Wert, wenn dies durch einen ersten der Prozessoren (22—24) angefordert wird;
während die Sperr-Flag auf den Gesperrt-Wert gesetzt ist, Verbieten des Zugriffes auf den Datenblock durch irgendeinen der Prozessoren und Anordnen einer Anfrage bezüglich eines Zugriffs auf den Datenblock oder einer Sperrung durch solche Prozessoren in einer Abfragewarteschlange (362) für ausgesetzte Zugriffe;
Rücksetzen der Sperr-Flag auf den Ungesperrt-Wert, wenn dies durch den ersten Prozessor abgefragt wird; und
wenn die Sperr-Flag auf den Ungesperrt-Wert zurückgesetzt ist, Verarbeiten der Anfragen in der Abfragewarteschlange für ausgesetzte Zugriffe.

8. Verfahren nach Anspruch 3, das ferner folgende Schritte aufweist:
wenn es durch einen Prozessor (22—24) angefragt wird, Durchführen einer globalen Reduktions-Operation bezüglich der Daten in dem globalen Speicher (32, 33), und Speichern eines Ergebnisses der globalen Reduktions-Operation an einer bestimmten Stelle des globalen Speichers.

9. Verfahren nach Anspruch 3, das ferner folgende Schritte aufweist:
Speichern einer Mehrzahl von globalen Kopien eines Datenblocks (118, 196, 280), in jedem einer Mehrzahl von globalen Speichern (32, 33);
Zuordnen einer Mehrzahl von Prozessoren (22—24) jedem der Mehrzahl von globalen Speichern;
wenn ein Prozessor (22—24) Zugriff auf den Datenblock anfragt, Zugreifen durch einen solchen Prozessor auf die globale Kopie des Datenblocks in einem globalen Speicher, das einem solchen Prozessor zugeordnet ist;
wobei der Schritt des Zugreifens auf die globale Kopie folgende Schritte aufweist;
Speichern einer lokalen Kopie (54, 86, 224) der globalen Kopie in einem lokalen Speicher (44—46) des Zugreifenden Prozessors; und
wenn die lokale Kopie aus dem lokalen Speicher gelöscht wird, Verbinden der lokalen Kopie mit der globalen Kopie in dem globalen Speicher, der dem zugreifenden Prozessor zugeordnet ist; und
nachdem alle lokalen Kopien einer der globalen Kopien mit einer solchen globalen Kopie verbunden sind, Verbinden solcher globalen Kopien mit weiteren globalen Kopien.

10. Multiprozessor-Computersystem (20) mit gemeinsam benutztem Speicher, mit:
einem oder mehreren globalen Speichereinheiten (26, 27), wobei jede globale Speichereinheit einen globalen Speicher zum Speichern einer Mehrzahl von Datenblöcken (118, 196, 280) der gemeinsam verwendeten Daten (32, 33) umfaßt, wobei jeder Datenblock eine Mehrzahl von Mehr-Bit-Datenelementen (120—122, 190—194, 296—299) umfaßt;
einer Mehrzahl von Verarbeitungseinheiten (22—24), die mit dem einen oder mehreren globalen Speichereinheiten verbunden sind, wobei jede der Verarbeitungseinheiten einen Prozessor (36—38) und einen lokalen Speicher (44—46) zum Speichern einer Schreibe-Bitmaske (56, 88, 220) und einer lokalen Kopie (54, 86, 224) des einen der Datenblöcke umfaßt, wenn Zugriff auf die Daten in dem Datenblock durch den Prozessor angefordert wird, wobei die Schreibe-Bitmaske eine Mehrzahl von Flags (60—64, 90—92, 240—244) umfaßt, die in einer 1 : 1-Beziehung den Datenelementen (70—74, 80—82, 230—234) der lokalen Kopie zugeordnet sind, wobei die Flags anfänglich auf einen Nicht-geschrieben-Wert gesetzt sind, wenn die lokale Kopie in dem lokalen Speicher gespeichert wird;
wobei jede der Verarbeitungseinheiten ferner eine lokale Speichersteuerung (40—42) umfaßt, wobei jede lokale Speichersteuerung wirksam ist, um diejenigen Schreibe-Bitmasken-Flags auf einen Geschrieben-Wert zu setzen, die Datenelementen, die durch den Prozessor geschrieben wurden, zugeordnet sind, und um, wenn die lokale Kopie mit anderen Daten in dem lokalen Speicher ersetzt wird, die Schreibe-Bitmaske und die geschriebenen Datenelemente an die globale Speichereinheit zu senden, wenn irgendwelche Datenelemente durch den Prozessor geschrieben wurden;
einer globalen Speichersteuerung (30, 31) in der globalen Speichereinheit, die wirksam ist, um eine lokale Kopie, die von einer der Verarbeitungseinheiten empfangen wurde durch Speichern jedes der Datenelemente im Datenblock, deren zugeordnete Schreibe-Bitmaske-Flag auf den Geschrieben-Wert gesetzt ist, zu verbinden.

Hierzu 6 Seite(n) Zeichnungen

– Leerseite –

Nummer: DE 44 23 559 A1
Int. Cl.[6]: G 06 F 12/08
Offenlegungstag: 11. Mai 1995

FIG. 1

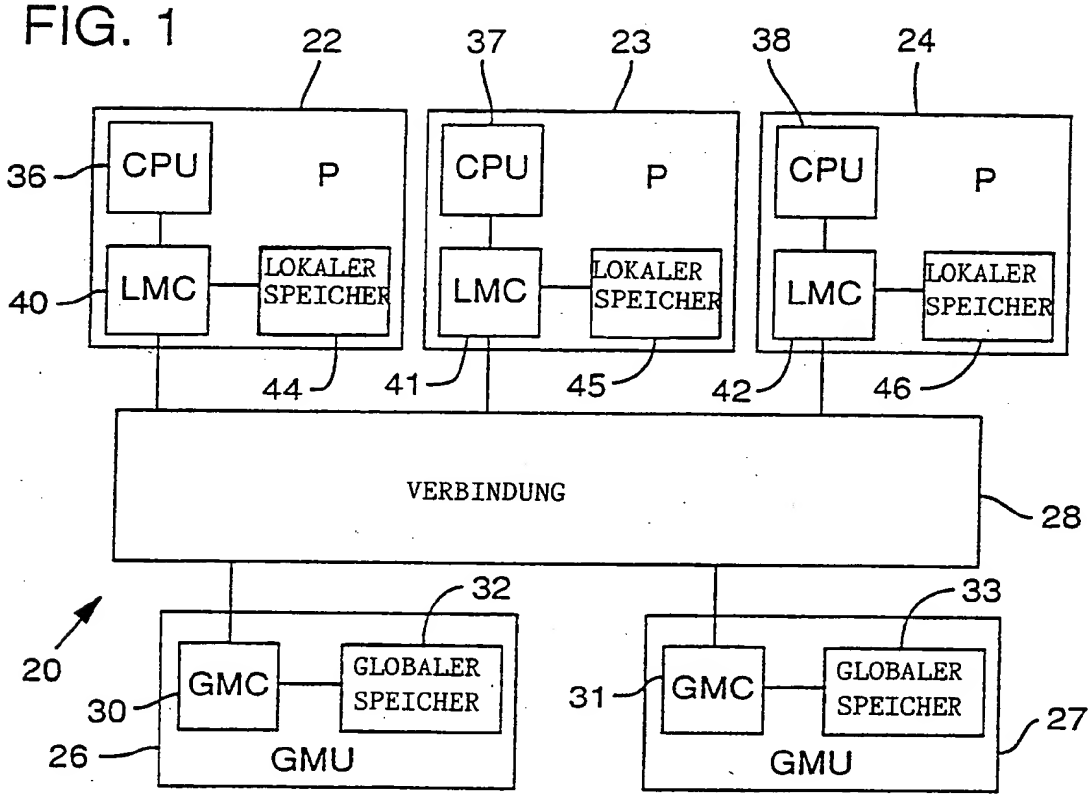
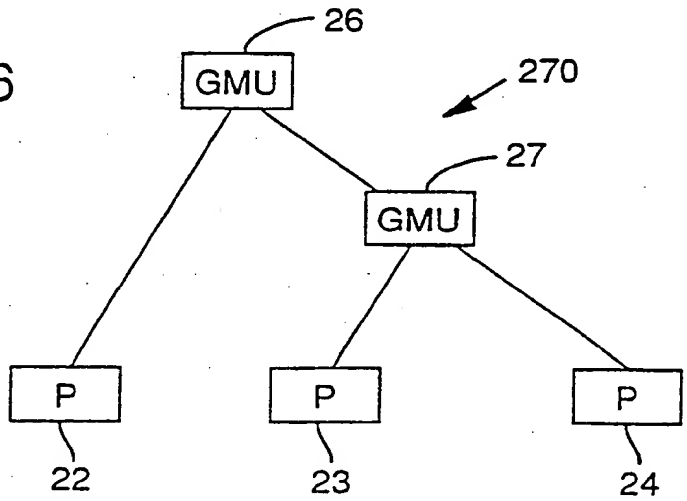


FIG. 6



Nummer: DE 44 23 559 A1
Int. Cl.[6]: G 06 F 12/08
Offenlegungstag: 11. Mai 1995

FIG. 2

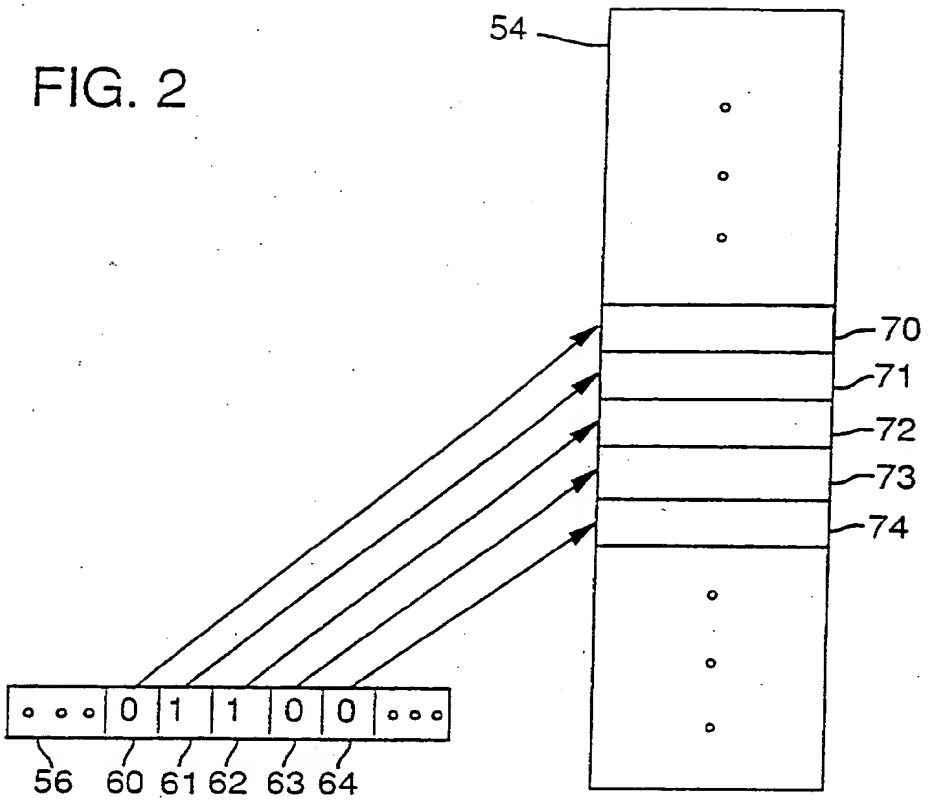
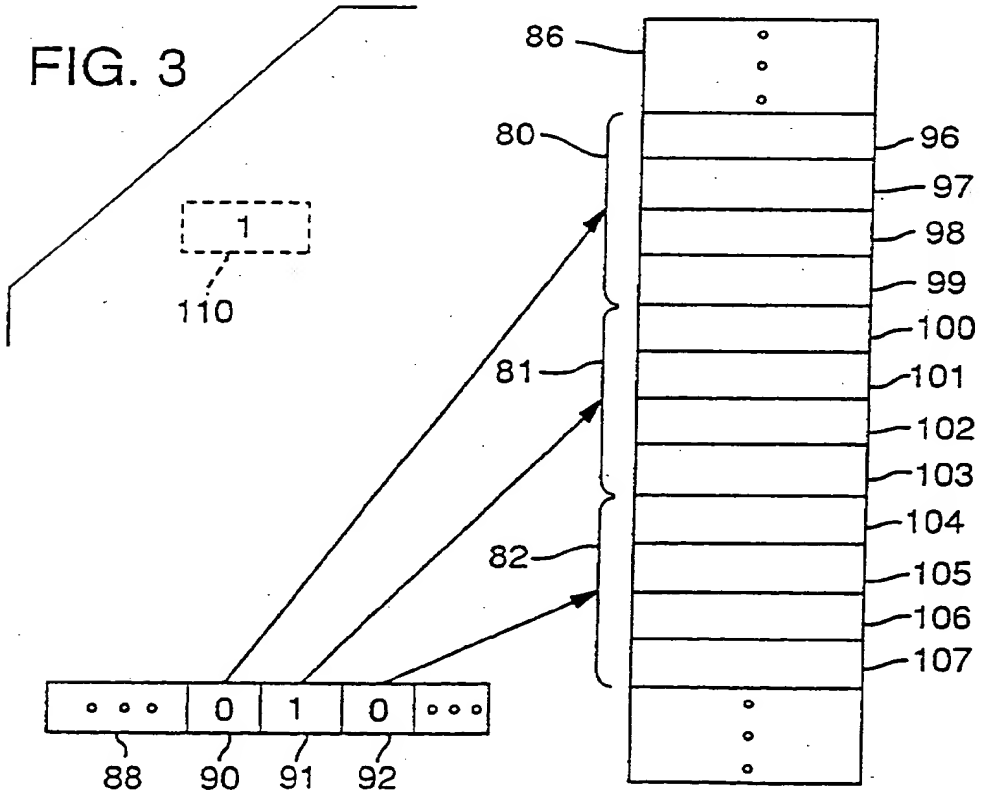
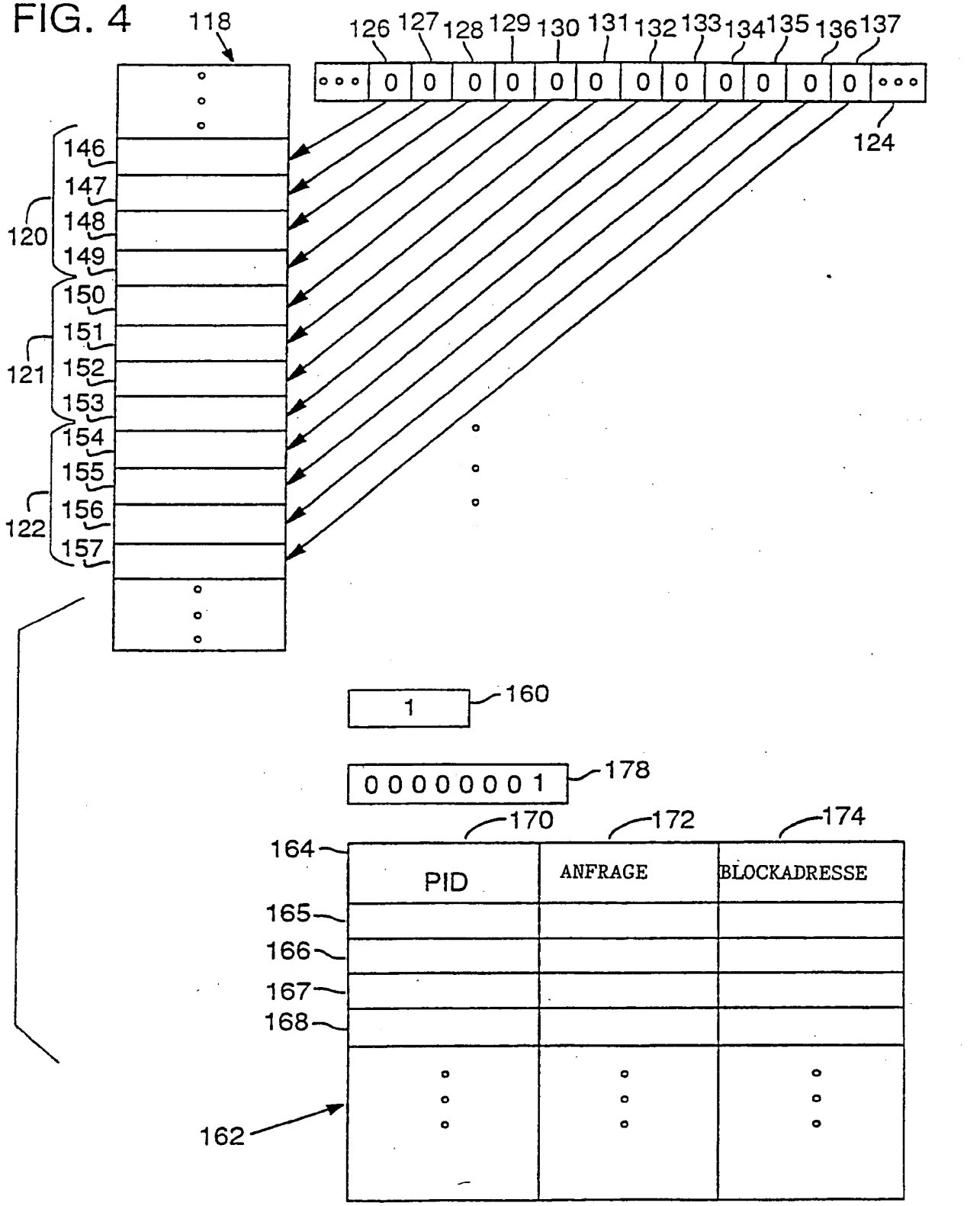


FIG. 3



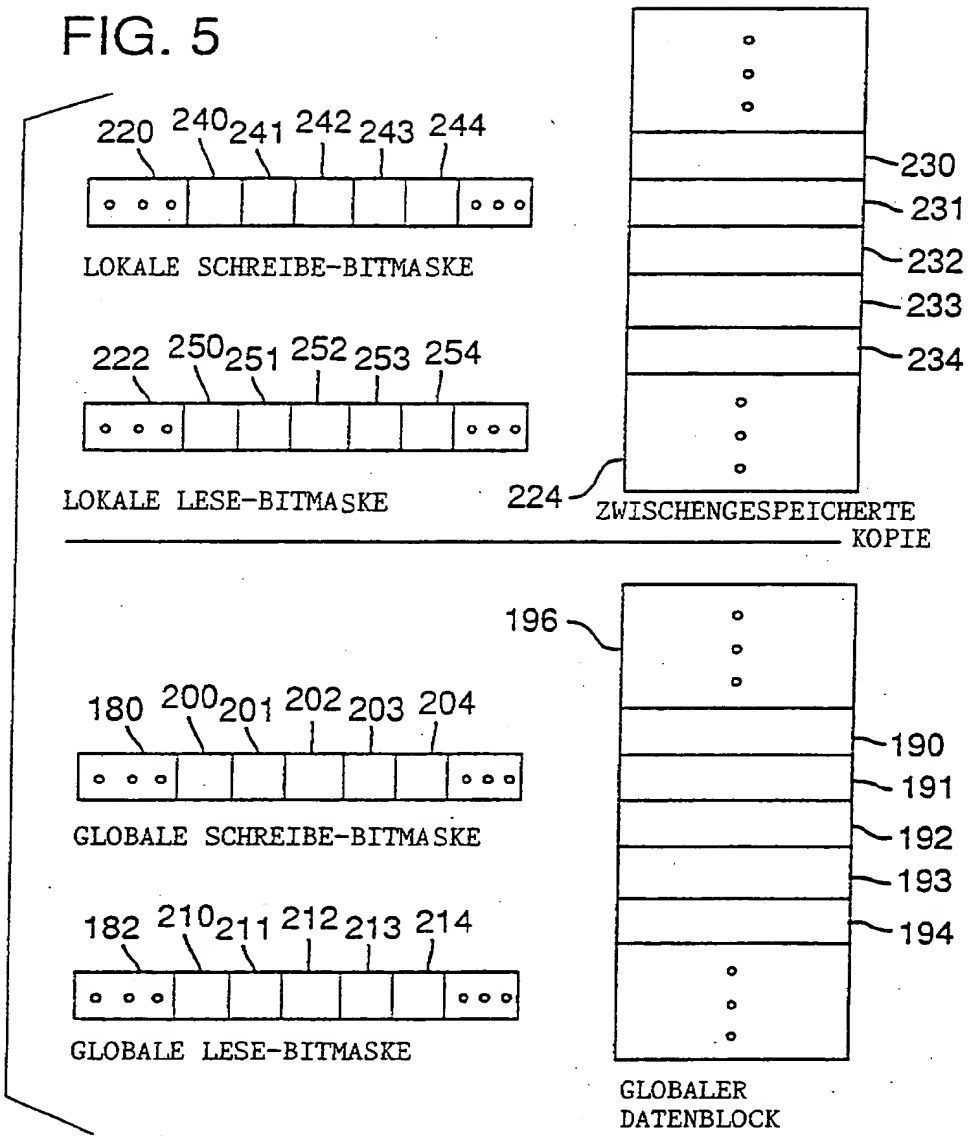
Nummer: DE 44 23 559 A1
Int. Cl.[6]: G 06 F 12/08
Offenlegungstag: 11. Mai 1995

FIG. 4



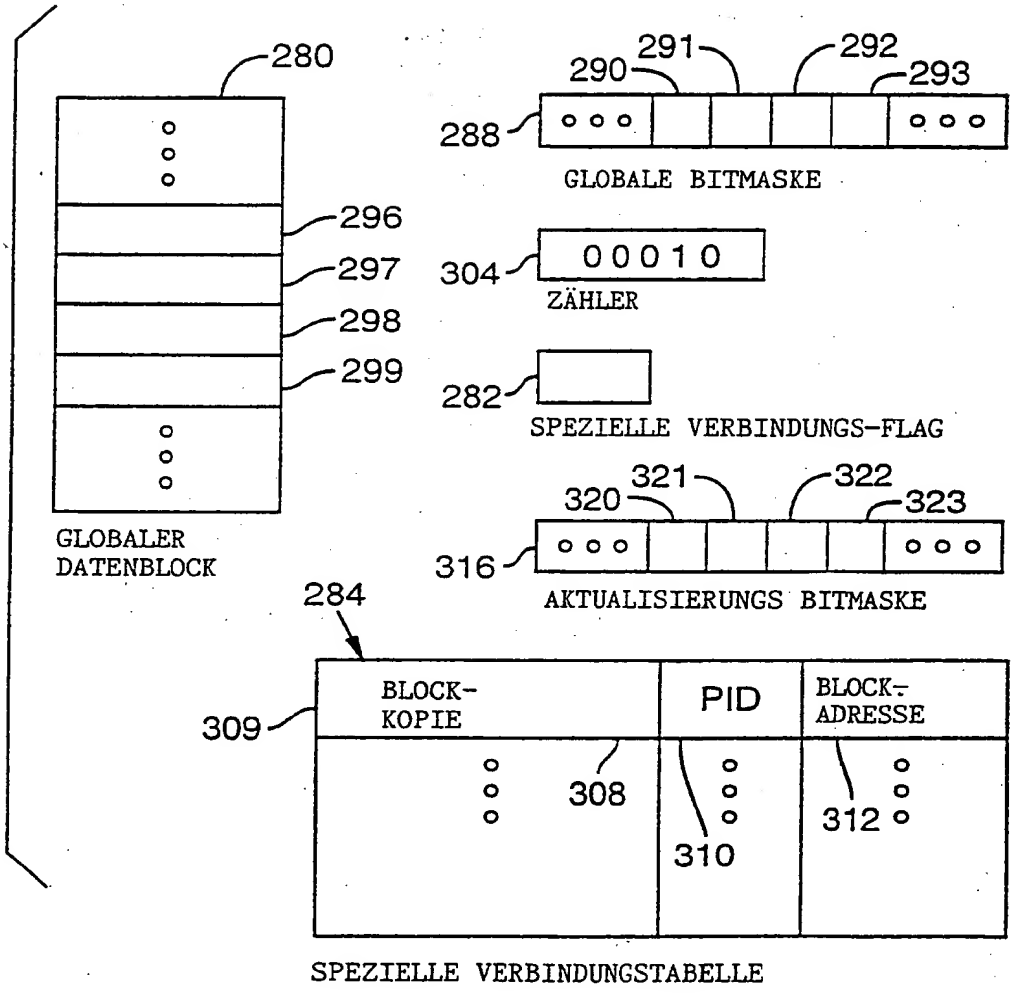
Nummer: DE 44 23 559 A1
Int. Cl.[6]: G 06 F 12/08
Offenlegungstag: 11. Mai 1995

FIG. 5



Nummer: DE 44 23 559 A1
Int. Cl.[6]: G 06 F 12/08
Offenlegungstag: 11. Mai 1995

FIG. 7



Nummer: DE 44 23 559 A1
Int. Cl.[6]: G 06 F 12/08
Offenlegungstag: 11. Mai 1995

FIG. 8